MS-DOS® Series

For Personal Computers Running the MS-DOS®

Version 1.0

Microsoft® Visual Basic™

Programming System for MS-DOS®

マイクロソフト ビジュアル ベーシック　プログラミング システム

リファレンス

Microsoft® VISUAL BASIC™

Microsoft® Visual Basic™

Programming System for MS-DOS®

マイクロソフト ビジュアル ベーシック　プログラミング システム

リファレンス

Microsoft® VISUAL BASIC™

ご注意

① このソフトウェアの著作権は、米国マイクロソフト社にあります。
② このソフトウェアおよびマニュアルの一部または全部を無断で使用、複製することはできません。
③ ソフトウェアは、コンピュータ 1台につき 1セット購入が原則になっております。
④ このソフトウェアおよびマニュアルは、本製品の使用許諾契約書のもとでのみ使用することができます。
⑤ このソフトウェアおよびマニュアルを運用した結果の影響については、いっさい責任を負いかねますのでご了承ください。

MS, Microsoft, MS-DOS は米国マイクロソフト社の登録商標です。
QuickBasic, Visual Basic, Windows は米国マイクロソフト社の商標です。

Microsoft Visual Basic for MS-DOS のマニュアルは、Microsoft Word for Windows Version 1.2A を使用して製作しました。

目次

はじめに xiii

第 1 部　機能一覧

関数，ステートメント，メソッド 3
プロパティ 8
イベント 13
オブジェクト 15

第 2 部　A-Z リファレンス

オプション ボタン コントロール 19
コマンド ボタン コントロール 20
コンボ ボックス コントロール 21
タイマー コントロール 22
テキスト ボックス コントロール 23
ディレクトリ リスト ボックス コントロール 24
チェック ボックス コントロール 25
ドライブ リスト ボックス コントロール 26
ピクチャー ボックス コントロール 27
ファイル リスト ボックス コントロール 28
フォーム 29
フレーム コントロール 31
メニュー コントロール 32
ラベル コントロール 34
リスト ボックス コントロール 35
水平，垂直スクロール バー コントロール 36
ABS 関数 37
Absolute ルーチン 37
ActiveControl, ActiveForm プロパティ 38
ADDITEM メソッド 39
Alignment プロパティ 40
Archive, Hidden, Normal, ReadOnly, System プロパティ 40
ASC, CHR$ 関数 41
ATN 関数 42
Attached プロパティ 42
AutoRedraw プロパティ 43

AutoSize プロパティ 44
BackColor, ForeColor プロパティ 45
BEEP ステートメント 45
BLOAD, BSAVE ステートメント 46
BorderStyle プロパティ 47
BSAVE ステートメント 49
CALL ステートメント (Basic プロシージャ) 49
CALL[S] ステートメント (Non-Basic プロシージャ) 50
Cancel プロパティ 51
Caption プロパティ 52
CDBL$ 関数 53
CHAIN ステートメント 54
Change イベント 55
CHDIR, MKDIR, RMDIR, FILES ステートメント 56
CHDRIVE ステートメント 57
Checked プロパティ 58
CHR$ 関数 58
CIRCLE ステートメント 58
CLEAR メソッド 60
CLEAR ステートメント 60
Click イベント 61
CLIPBOARD オブジェクト 62
CLOSE ステートメント 63
CLS メソッド 64
CLS ステートメント 64
COLOR ステートメント 65
COM ステートメント 66
COMMAND$ 関数 67
COMMON ステートメント 68
CONST ステートメント 72
ControlBox プロパティ 73
ControlPanel プロパティ 74
COS, SIN, TAN 関数 76
CSNG$ 関数 77
CSRLIN 関数 77
CtlName プロパティ 78
C*type* 系関数 (CCUR, CINT, CLNG, CSNG, CDBL) 79
CURDIR$ 関数 80
CurrentX, CurrentY プロパティ 81
CV*x* 系関数 (CVC, CVI, CVL, CVS, CVD) 82

CV*x*MBF 系, MK*x*MBF$ 系関数
(CVSMBF, CVDMBF, MKSMBF$, MKDMBF$) 82
DATA, READ, RESTORE ステートメント 83
DATE$ 関数, DATE$ ステートメント 85
DATESERIAL 関数 86
DATEVALUE 関数 86
DAY 関数 87
DblClick イベント 87
DECLARE ステートメント (Basic プロシージャ) 88
DECLARE ステートメント (Non-Basic プロシージャ) 90
DEF FN ステートメント 92
DEF SEG ステートメント 94
Default プロパティ 95
DEF*type* 系ステートメント 96
DIM ステートメント 97
DIR$ 関数 99
DO...LOOP ステートメント 100
DOEVENTS 関数 101
DRAG メソッド 102
DragDrop イベント 102
DragMode プロパティ 104
DragOver イベント 105
DRAW ステートメント 106
Drive プロパティ 108
DropDown イベント 109
$DYNAMIC, $STATIC メタコマンド 110
Enabled プロパティ 111
END ステートメント 112
ENDDOC メソッド 113
ENVIRON ステートメント, ENVIRON$ 関数 113
EOF 関数 114
ERASE ステートメント 115
ERDEV, ERDEV$ 関数 116
ERR ステートメント 117
ERR, ERL 関数 118
ERROR ステートメント 119
ERROR$ 関数 119
EVENT ステートメント 120
EXIT ステートメント 120
EXP, LOG 関数 121

FIELD ステートメント 122
FILEATTR 関数 123
FileName プロパティ 123
FILES ステートメント 124
FIX, INT 関数 124
FOR...NEXT ステートメント 125
ForeColor プロパティ 126
$FORM メタコマンド 127
FORMAT$ 関数 127
FormName プロパティ 130
FormType プロパティ 131
FRE 関数 132
FREEFILE 関数 134
FUNCTION ステートメント 134
GET, PUT ステートメント (ファイル I/O) 137
GET, PUT ステートメント (グラフィックス) 139
GetFEPMode% ルーチン 141
GETTEXT メソッド 142
GOSUB...RETURN ステートメント 142
GotFocus イベント 144
GOTO ステートメント 145
Height, Width プロパティ 145
HEX$, OCT$ 関数 147
Hidden プロパティ 147
HIDE メソッド 147
HOUR 関数 148
IF TYPEOF ステートメント 149
IF...END IF ステートメント 150
$INCLUDE メタコマンド 152
Index プロパティ 153
INKEY$ 関数 154
INP 関数, OUT ステートメント 155
INPUT, LINE INPUT ステートメント 155
INPUT #, LINE INPUT # ステートメント 157
INPUT$ 関数 158
INPUT¥ 関数 159
INPUTBOX$ 関数 159
INSTR 関数 160
INT 関数 161
Interrupt, InterruptX ルーチン 161

Interval プロパティ 163
IOCTL$ 関数, IOCTL ステートメント 163
JIS$ 関数 164
KEXT$ 関数 165
KEY ステートメント (イベント トラッピング) 165
KEY ステートメント (代入) 168
KeyDown, KeyUp イベント 169
KeyPress イベント 170
KILL ステートメント 172
KINSTR 関数 172
KLEN 関数 173
KMID$ 関数 173
KMID$ ステートメント 174
KPOS 関数 175
KTN$ 関数 175
LargeChange, SmallChange プロパティ 176
LBOUND, UBOUND 関数 177
LCASE$, UCASE$ 関数 178
Left, Top プロパティ 178
LEFT$, RIGHT$ 関数 179
LEN 関数 180
LET ステートメント 180
LINE ステートメント 181
LINE INPUT ステートメント 182
LINE INPUT # ステートメント 182
List プロパティ 182
ListCount プロパティ 184
ListIndex プロパティ 185
Load イベント 187
LOAD ステートメント 187
LOC 関数 189
LOCATE ステートメント 189
LOCK, UNLOCK ステートメント 190
LOF 関数 191
LOG 関数 191
LostFocus イベント 192
LPOS 関数 193
LPRINT ステートメント 193
LSET, RSET ステートメント 194
LTRIM$, RTRIM$ 関数 195

Max, Min プロパティ 196
MaxButton プロパティ 197
MID$ 関数, MID$ ステートメント 198
Min プロパティ 198
MinButton プロパティ 199
MINUTE 関数 199
MKDIR ステートメント 199
MKx$ 系関数 (MKC, MKI, MKL, MKS, MKD) 200
MKxMBF$ 系関数 (MKSMBF$, MKDMBF$) 200
MONTH 関数 200
MouseDown, MouseUp イベント 201
MouseMove イベント 202
MousePointer プロパティ 203
MouseUp イベント 204
MOVE メソッド 205
MSGBOX ステートメント, MSGBOX 関数 206
MultiLine プロパティ 208
NAME ステートメント 208
NEWPAGE メソッド 209
Normal プロパティ 209
NOW 関数 210
OCT$ 関数 210
ON ERROR ステートメント 210
ON *event* 系ステートメント
(ON {COM|KEY|PLAY|TIMER|UEVENT}) 212
ON...GOSUB, ON...GOTO ステートメント 216
OPEN ステートメント (ファイル I/O) 217
OPEN COM ステートメント 220
OPTION BASE ステートメント 224
OPTION EXPLICIT ステートメント 225
OUT ステートメント 225
Paint イベント 226
PAINT ステートメント 227
PALETTE, PALETTE USING ステートメント 228
Parent プロパティ 230
Path プロパティ 231
PathChange イベント 232
Pattern プロパティ 232
PatternChange イベント 233
PCOPY ステートメント 233

PEEK 関数, POKE ステートメント 234
PLAY ステートメント (イベント トラッピング) 235
PLAY ステートメント (ミュージック) 236
PLAY 関数 238
PMAP 関数 239
POINT 関数 239
POKE ステートメント 240
POS 関数, CSRLIN 関数, LOCATE ステートメント 240
PRESET, PSET ステートメント 242
PRINT メソッド 243
PRINT ステートメント 244
PRINT # ステートメント 247
PRINTER オブジェクト 248
PRINTFORM メソッド 249
PrintTarget プロパティ 249
PSET ステートメント 250
PUT ステートメント (ファイル I/O) 250
PUT ステートメント (グラフィックス) 250
RANDOMIZE ステートメント, RND 関数 250
READ ステートメント 251
ReadOnly プロパティ 251
REDIM ステートメント 252
REFRESH メソッド 254
REM ステートメント 254
REMOVEITEM メソッド 255
RESET ステートメント 255
Resize イベント 256
RESTORE ステートメント 256
RESUME ステートメント 256
RGB 関数 257
RIGHT$ 関数 258
RMDIR ステートメント 258
RND 関数 258
RSET ステートメント 258
RTRIM$ 関数 258
RUN ステートメント 259
SADD 関数 260
ScaleHeight, ScaleWidth プロパティ 260
SCREEN オブジェクト 261
SCREEN ステートメント 262

SCREEN 関数 263
ScrollBars プロパティ 264
SECOND 関数 264
SEEK 関数, SEEK ステートメント 265
SELECT CASE ステートメント 266
SelLength, SelStart, SelText プロパティ 268
Separator プロパティ 269
SetFEPMode ルーチン 270
SETFOCUS メソッド 270
SETMEM 関数 271
SETTEXT メソッド 272
SetUEvent ルーチン 272
SGN 関数 273
SHARED ステートメント 273
SHELL ステートメント 274
SHOW メソッド 274
SIN 関数 275
SLEEP ステートメント 276
SmallChange プロパティ 276
Sorted プロパティ 276
SOUND ステートメント 277
SPACE$ 関数 278
SPC 関数 278
SQR 関数 279
SSEG 関数 279
SSEGADD 関数 280
STACK ステートメント 280
STACK 関数 281
$STATIC メタコマンド 281
STATIC ステートメント 281
STOP ステートメント 282
STR$, VAL 関数 283
STRING$ 関数 283
StringAddress ルーチン 284
StringAssign ルーチン 284
StringLength ルーチン 286
StringRelease ルーチン 287
Style プロパティ 288
SUB ステートメント 289
SWAP ステートメント 291

System プロパティ 292
SYSTEM ステートメント 292
TAB 関数 293
TabIndex プロパティ 294
TabStop プロパティ 294
Tag プロパティ 295
TAN 関数 296
Text プロパティ 296
TEXTHEIGHT, TEXTWIDTH メソッド 297
TIME$ 関数, TIME$ ステートメント 298
Timer イベント 298
TIMER ステートメント 299
TIMER 関数 300
TIMESERIAL 関数 300
TIMEVALUE 関数 300
Top プロパティ 301
TRON, TROFF ステートメント 301
TYPE ステートメント 302
UBOUND 関数 303
UCASE$ 関数 303
UEVENT ステートメント 304
Unload イベント 305
UNLOAD ステートメント 305
UNLOCK ステートメント 306
VAL 関数 306
Value プロパティ 306
VARPTR$ 関数 307
VARPTR, VARSEG 関数 308
VARSEG 関数 309
VIEW PRINT ステートメント 309
VIEW ステートメント 310
Visible プロパティ 311
WAIT ステートメント 312
WEEKDAY 関数 312
WHILE...WEND ステートメント 313
Width プロパティ 313
WIDTH ステートメント 314
WINDOW ステートメント 315
WindowState プロパティ 316
WRITE ステートメント 316

WRITE # ステートメント 317
YEAR 関数 318

付録

A 予約語一覧 321

B データ型 325
INTEGER (整数型) 325
LONG (長整数型) 326
SINGLE (単精度浮動小数点数型) 326
DOUBLE (倍精度浮動小数点型) 326
CURRENCY (通貨型) 326
STRING (文字列型) 327
CONTROL (コントロール型) 327
FORM (フォーム型) 327

C 演算子 329
AND 演算子 330
EQV 演算子 330
IMP 演算子 330
NOT 演算子 331
OR 演算子 331
XOR 演算子 331

D エラー メッセージ一覧 333
D.1 トラップ可能な実行時エラー 333
D.2 BC コンパイラのエラー 336
D.3 LIB のエラー 355
LIB の致命的なエラー 355
LIB のエラー 359
LIB の警告 360
D.4 LINK のエラー 361
LINK の致命的なエラー 361
LINK のエラー 373
LINK の警告 377

はじめに

Visual Basic for MS-DOS のマニュアル構成

Visual Basic for MS-DOS のパッケージには、次の 3 冊のマニュアルと、オンライン ヘルプが含まれています。

- 『プログラミング ガイド』

  Visual Basic for MS-DOS を使ってどのようにアプリケーションを作成するかを、例をまじえながら説明します。実際に簡単なアプリケーションを作成しながら、プログラミング環境でコードを記述する方法と、フォーム デザイナでフォームを設計，作成する方法を説明した後、Visual Basic for MS-DOS がサポートしている各機能をどのように活用していくかを順に触れて行きます。このマニュアルを読んで、コードの動作と意味について理解してください。

- 『リファレンス』(このマニュアル)

  Visual Basic for MS-DOS の言語仕様に関するリファレンスです。

- 『機種依存情報』

  グラフィックスやキーボード スキャン コードなどの、ハードウェアに依存する機能について説明しています。例えば、グラフィックスを表示するときに必要なスクリーン モードの種類と値などは、このマニュアルに載っています。『プログラミング ガイド』および『リファレンス』には、ハードウェアに依存した機能の詳細は記載されておらず、このマニュアルを参照するようになっています。また、Microsoft の以前のバージョンの Basic との互換性についても、このマニュアルを参照してください。

- オンライン ヘルプ

  Visual Basic for MS-DOS の操作方法と言語仕様についてのリファレンスです。マニュアルを読んでVisual Basic の基本的な内容について理解した後は、構文などの細かい仕様やメニュー コマンドの機能を調べるために、このオンライン ヘルプを利用してください。オンライン ヘルプのトピックを表示するには、キーワードにカーソルを合わせて [F1] キーを押します。また、ダイアログ ボックスなどで <ヘルプ> ボタンがあれば、より詳しい説明を含むトピックが用意されています。ほとんどの関数やステートメントの説明には、プログラム例が含まれます。これらの例は、コード ウィンドウに複写/貼り付けして実行でき、アプリケーションで利用することができます。

このマニュアルの構成

このマニュアルは 3 部構成になっており、それぞれ次のような内容について解説しています。

- 第 1 部「機能一覧」では、Visual Basic for MS-DOS がサポートしている関数，ステートメント，メソッド，プロパティ，イベント，オブジェクトを機能別に要約し、それぞれのオブジェクトに適用可能なメソッド，プロパティ，イベントの対応表を示してあります。例えば、あるオブジェクトはどのようなプロパティを持っているかを調べるときに便利です。
- 第 2 部「A-Z リファレンス」では、関数，ステートメント，メソッド，プロパティ，イベント，オブジェクトの詳細について、アルファベット順に説明しています。
- 付録には、予約語の一覧表、データ型と演算子の解説、トラップ可能なエラー メッセージの一覧などが含まれています。

プログラム例は、オンライン ヘルプを参照してください。オンライン ヘルプには、このマニュアルと同等の情報が含まれています。また、パッケージに含まれている README.TXT ファイルも必ず参照するようにしてください。

マニュアルの表記規則

このマニュアルは、次の表記規則に従って記述されています。

表記	説明
SUB, IF, CHDIR, PRINT, TIME$	大文字のボールド (太字) は、Visual Basic for MS-DOS の予約語であることを表します。
BackColor, Click	プロパティとイベントの名前は、大文字で始まります。
setup	プログラマが実際に入力する単語はボールドで示します。
eventname	構文中のイタリック (斜体) は、ファイル名のように適当な文字列に置き換える必要のあるものを表します。単語の最後の文字は、指定するデータ型を表すこともあります。
[*expressionlist*]	構文中の角カッコ ([]) は、囲まれた項目が省略できることを表します。
{**While**|**Until**}	構文中の大カッコ ({}) と縦棒の組み合わせは、複数の項目から選択する必要があることを表します。全体が角カッコ ([]) で囲まれていなければ、いずれか 1 つの項目を選ばなければなりません。

`SUB HelloButton_Click ()`
`  Readout.Text = "Hello, world!"`
`END SUB`
	この書体は、変数，コード，例，エラーメッセージを表します。

`Readout.Text = "This should all be ⇒ on one line in the Code window."`	行連結文字 (⇒) はコードが次の行に続くことを表します。実際にプログラムとして入力するときは、1 行で入力します。
CONSTANT.BI, F1_KEY	ファイル名や定数名は、すべて大文字で示してあります。
[ENTER]	説明文中の角カッコ ([]) は、キーボード上のキーやメニュー項目，ダイアログ ボックスの名前などを表します。Visual Basic for MS-DOS のマニュアルやオンライン ヘルプでは、キーの表記にジェネリック表記を採用しています。一部のキーは、キーボード上に書かれた表記と異なりますから注意してください。例えばリターン キーは [ENTER]，メニューをアクティブにするためのキーは [メニュー] キーです。 Visual Basic for MS-DOS でのジェネリック表記は、Microsoft Windows 用のアプリケーションと合わせているため、Microsoft Basic や QuickBasic での表記とは異なります。 ジェネリック表記と実際のキーボード上の表記との対応は、『機種依存情報』またはオンライン ヘルプを参照してください。
[メニュー] + [F1]	プラス記号 (+) は、キーの組み合わせを表します。例えば [メニュー] + [F1] は、[メニュー] キーを押しながら [F1] キーを押すことを表します。
Multiple Document Interface (MDI)	略語はその用語を最初に使うときは正式名称を書きます。

第 1 部
機能一覧

関数，ステートメント，メソッド

表1は、プログラムのタスクごとに、関連する関数，ステートメント，メソッドをまとめたものです。また表2は、メソッドと、そのメソッドを適用可能なオブジェクトをアルファベット順に示してあります。

表1　プログラム タスクごとの関数、ステートメント，メソッド一覧

カテゴリ	機能	関数、ステートメント，メソッド
配列	配列変数の宣言と初期化	DIM, REDIM, STATIC
	配列変数の再初期化	ERASE, REDIM, REDIM PRESERVE
	添字の範囲	LBOUND, UBOUND
	添字のデフォルトの最小値の設定	OPTION BASE
プログラムの流れの制御	ループ	DO...LOOP, FOR...NEXT, WHILE...WEND
	分岐	GOSUB...RETURN, GOTO, ON ERROR, ON...GOSUB, ON...GOTO
	条件判断	IF...ENDIF, IFTYPEOF, SELECT CASE
	プログラムの終了，中断	DOEVENTS, END, STOP, UNLOAD, SYSTEM, CHAIN, RUN
変換	ASCII を文字列に	CHR$
	文字列を ASCII に	ASC
	数値を文字列に	STR$, FORMAT$
	文字列を数値に	VAL
	数値データ型同士	CCUR, CDBL, CINT, CLNG, CSNG, FIX, INT
	8 進, 16 進文字列に	HEX$, OCT$
	日付をシリアル値に	DATESERIAL, DATEVALUE
	シリアル値を日付に	DAY, MONTH, WEEKDAY, YEAR, NOW
	時刻をシリアル値に	TIMESERIAL, TIMEVALUE
	シリアル値を時刻に	HOUR, MINUTE, SECOND, NOW
カット，コピー，ペースト	CLIPBOARD オブジェクトを使って	CLEAR, GETTEXT, SETTEXT

表1　プログラム タスクごとの関数，ステートメント，メソッド一覧（続き）

カテゴリ	機能	関数，ステートメント，メソッド
日付，時刻	取得	DATE$, NOW, TIME$
	設定	DATE$, TIME$
	経過時間	TIMER
グラフィックス	色	POINT, RGB
	図形の描画	LINE, CIRCLE, PSET
	テキストのサイズ	TEXTHEIGHT, TEXTWIDTH
	画面イメージ	GET, PUT
	テキストの表示	PRINT
	オブジェクト，画面，ビューポートのクリア	CLS
エラー トラップ	実行時のエラーのトラップ	ON ERROR, RESUME
	エラー状況の取得	ERR, ERL
	エラー メッセージの取得	ERROR$
	疑似的にエラーを発生	ERROR
ファイル入出力	ファイルのオープン	OPEN
	ファイルのクローズ	CLOSE, RESET
	ファイルへの書き込み	PRINT #, PUT, WRITE #
	出力の体裁の制御	SPC, TAB, WIDTH#
	ファイルからの読み出し	GET, INPUT #, INPUT$, INPUT¥, LINE INPUT #
	ファイルに関する情報の取得	EOF, FILEATTR, FREEFILE, LOC, LOF, SEEK
	操作位置の設定	SEEK
	ディスクやディレクトリの操作	CHDRIVE, CURDIR$, CHDIR, MKDIR, RMDIR
	ファイルの操作	DIR$, KILL, LOCK...UNLOCK, NAME

表 1　プログラム タスクごとの関数，ステートメント，メソッド一覧（続き）

カテゴリ	機能	関数，ステートメント，メソッド
オブジェクト操作	フォームの表示/非表示	SHOW, HIDE
	オブジェクトのロード/アンロード	LOAD, UNLOAD
	オブジェクトの移動，サイズ変更	MOVE
	ドラッグ & ドロップ	DRAG
	ユーザー入力対象のコントロールの設定	SETFOCUS
	表示の更新	REFRESH
	フォームの印刷	PRINTFORM
	リスト ボックス，コンボ ボックスの項目の操作	ADDITEM, REMOVEITEM
	ダイアログ ボックスの表示	INPUTBOX$, MSGBOX
演算	一般的な計算	EXP, LOG, SQR
	三角関数	ATN, COS, SIN, TAN
	数値の変換	FIX, INT
	絶対値の取得	ABS
	符号の取得	SGN
	乱数の発生	RANDOMIZE, RND
印刷	プリンタの制御	ENDDOC, NEWPAGE
	出力の体裁の制御	SPC, TAB, TEXTHEIGHT, TEXTWIDTH
	印刷	PRINT, PRINTFORM, LPRINT
プロシージャ	プロシージャの宣言	FUNCTION, SUB
	SUB プロシージャの呼び出し	CALL
	プロシージャの終了	EXIT FUNCTION, EXIT SUB, END FUNCTION, END SUB
	外部のプロシージャの宣言	DECLARE

表1　プログラム タスクごとの関数，ステートメント，メソッド一覧（続き）

カテゴリ	機能	関数，ステートメント，メソッド
文字列	書式化	FORMAT$
	文字を繰り返した文字列	SPACE$, STRING$
	ASCII コード変換	ASC, CHR$
	文字列操作	INSTR, KINSTR, LEFT$, LTRIM$, MID$, KMID$, RIGHT$, RTRIM$, KEXT$
	大文字/小文字変換	LCASE$, UCASE$
	右寄せ/左寄せ	LSET, RSET
	文字列の長さ	LEN, KLEN, KPOS
	1バイト/2バイト変換	CDBL$, CSNG$
	文字コード取得	JIS$, KTN$
変数と定数	変数，定数の宣言	CONST, DIM, STATIC, SHARED, COMMON
	値の代入	LET
	デフォルトのデータ型	DEF*type*
	ユーザー定義型の定義	TYPE
その他	他のプログラムの実行	SHELL, RUN, CHAIN
	イベントの処理	DOEVENTS
	コマンドライン引数の取得	COMMAND$
	環境変数	ENVIRON, ENVIRON$
	ビープ音	BEEP

表２　メソッドとオブジェクトの対応

メソッド＼オブジェクト	フォーム	ラベル	コマンドボタン	オプションボタン	チェックボックス	テキストボックス	リストボックス	コンボボックス	ピクチャーボックス	フレーム	タイマー	水平／垂直スクロールバー	ディレクトリリストボックス	ドライブリストボックス	ファイルリストボックス	メニュー	CLIPBOARD	PRINTER	SCREEN
ADDITEM							○	○											
CLEAR																	○		
CLS	○								○										
DRAG		○	○	○	○	○	○	○	○	○		○	○	○	○				
ENDDOC																		○	
GETTEXT																	○		
HIDE	○																		○
MOVE	○	○	○	○	○	○	○	○	○	○		○	○	○	○				
NEWPAGE																		○	
PRINT	○								○									○	
PRINTFORM	○																		
REFRESH	○	○	○	○	○	○	○	○	○	○		○	○	○	○				
REMOVEITEM							○	○											
SETFOCUS	○		○	○	○	○	○	○	○			○	○	○	○				
SETTEXT																	○		
SHOW	○																		○
TEXTHEIGHT	○								○										
TEXTWIDTH	○								○										

プロパティ

表 3 は、プログラムのタスクごとにプロパティを分類したものです。また表 4 は、プロパティと、そのプロパティを適用可能なオブジェクトをアルファベット順に示してあります。

表 3　プログラム タスクごとのプロパティ一覧

カテゴリ	機能	プロパティ
体裁/外観	色の変更	BackColor, ForeColor
	境界線の変更	BorderStyle
	マウスポインタの変更	MousePointer
	表示/非表示	Visible
ファイル	ファイル名の表示	Drive, FileName, Path, Pattern
	属性によるファイルの表示/非表示	Archive, Hidden, Normal, ReadOnly, System
フォームとコントロール	デフォルトのボタンの設定	Cancel, Default
	タイマー間隔の設定	Interval
	親フォームの確認	Parent
	スクロール バーの表示	ScrollBars (テキスト ボックスのみ)
	コンボ ボックスのタイプの指定	Style
	タブ オーダーの設定	TabIndex, TabStop
	コントロールの状態	Value
	コントロールの参照	CtlName, Index
	フォーカスを持つコントロール	ActiveControl
	リスト ボックスの操作	List, ListCount, ListIndex, Sorted
	スクロール	Max, Min, LargeChange, SmallChange
	チェック/未チェック	Checked
	イベントへの応答	Enabled
	コントロール メニュー	ControlBox
	フォームの最小化/最大化	MaxButton, MinButton, WindowState
	フォームの参照	FormName
	ドラッグ & ドロップ	DragMode

表3　プログラム タスクごとのプロパティ一覧（続き）

カテゴリ	機能	プロパティ
画面状態	コントロールの再描画	AutoRedraw
	自動サイズ変更	AutoSize
	位置	CurrentX, CurrentY
サイズと位置	位置の設定	Left, Top
	サイズの設定	Height, Width
テキスト	寄せ	Alignment
	自動サイズ変更	AutoSize
	表示	Caption, Text
	選択	SelLength, SelStart, SelText

表4　プロパティとオブジェクトの対応

プロパティ＼オブジェクト	フォーム	ラベル	コマンドボタン	オプションボタン	チェックボックス	テキストボックス	リストボックス	コンボボックス	ピクチャーボックス	フレーム	タイマー	水平／垂直スクロールバー	ディレクトリリストボックス	ドライブリストボックス	ファイルリストボックス	メニュー	CLIPBOARD	PRINTER	SCREEN
ActiveControl																			■
ActiveForm																			■
Alignment		○																	
Archive															○				
Attached												□							
AutoRedraw	○								○										
AutoSize		○																	
BackColor	○	○	○	○	○	○	○	○	○	○			○	○	○				
BorderStyle	□	○				□			○										
Cancel			○																
Caption	○	○	○	○	○					○						○			
Checked																○			
ControlBox	□																		
ControlPanel																			●
CtlName		△	△	△	△	△	△	△	△	△	△	△	△	△	△	△			
CurrentX	●								●										
CurrentY	●								●										
Default			○																
DragMode		○	○	○	○	○	○	○	○	○		○	○	○	○				
Drive														●					
Enabled	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○			
FileName															●				
ForeColor	○	○		○	○	○	○	○	○	○			○	○	○				
FormName	△																		
FormType	△																		
Height	○	○	○	○	○	○	○	○	○	○		○	○	○	○				■

○　デザイン時に設定可能。実行時に取得，設定ともに可能。
●　デザイン時には使用できない。実行時に取得，設定ともに可能。
□　デザイン時に設定可能。実行時に取得のみ可能。
■　デザイン時には使用できない。実行時に取得または参照のみ可能。
△　デザイン時に設定可能。実行時には変更できない。
▲　デザイン時に設定可能。実行時には意味を持たない。

表 4　プロパティとオブジェクトの対応（続き）

プロパティ＼オブジェクト	フォーム	ラベル	コマンドボタン	オプションボタン	チェックボックス	テキストボックス	リストボックス	コンボボックス	ピクチャーボックス	フレーム	タイマー	水平／垂直スクロールバー	ディレクトリリストボックス	ドライブリストボックス	ファイルリストボックス	メニュー	CLIPBOARD	PRINTER	SCREEN
Hidden															○				
Index		△	△	△	△	△	△	△	△	△	△	△	△	△	△	△			
Interval											○								
LargeChange												○							
Left	○	○	○	○	○	○	○	○	○	○	▲	○	○	○	○				
List							●	●					■	■	■				
ListCount							■	■					■	■	■				
ListIndex							●	●					●	●	●				
Max												○							
MaxButton	□																		
Min												○							
MinButton	□																		
MousePointer	○	○	○	○	○	○	○	○	○	○		○	○	○	○				●
MultiLine						□													
Normal															○				
Parent	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■			
Path													●		●				
Pattern															○				
PrintTarget																		●	
ReadOnly															○				
ScaleHeight	■								■										
ScaleWidth	■								■										
ScrollBars						□													
SelLength						●		●											
SelStart						●		●											
SelText						●		●											

○　デザイン時に設定可能。実行時に取得，設定ともに可能。
●　デザイン時には使用できない。実行時に取得，設定ともに可能。
□　デザイン時に設定可能。実行時に取得のみ可能。
■　デザイン時には使用できない。実行時に取得または参照のみ可能。
△　デザイン時に設定可能。実行時には変更できない。
▲　デザイン時に設定可能。実行時には意味を持たない。

表 4　プロパティとオブジェクトの対応 (続き)

プロパティ＼オブジェクト	フォーム	ラベル	コマンドボタン	オプションボタン	チェックボックス	テキストボックス	リストボックス	コンボボックス	ピクチャーボックス	フレーム	タイマー	水平／垂直スクロールバー	ディレクトリリストボックス	ドライブリストボックス	ファイルリストボックス	メニュー	CLIPBOARD	PRINTER	SCREEN
Separator																○			
SmallChange												○							
Sorted							□	□											
Style								□											
System															○				
TabIndex		○	○	○	○	○	○	○	○	○		○	○	○	○				
TabStop			○	○	○	○	○	○	○			○	○	○	○				
Tag	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○			
Text						○	■	○											
Top	○	○	○	○	○	○	○	○	○	○	▲	○	○	○	○				
Value			●	○	○							○							
Visible	○	○	○	○	○	○	○	○	○	○		○	○	○	○	○			
Width	○	○	○	○	○	○	○	○	○	○		○	○	○	○				●
WindowState	○																		

○　デザイン時に設定可能。実行時に取得，設定ともに可能。
●　デザイン時には使用できない。実行時に取得，設定ともに可能。
□　デザイン時に設定可能。実行時に取得のみ可能。
■　デザイン時には使用できない。実行時に取得または参照のみ可能。
△　デザイン時に設定可能。実行時には変更できない。
▲　デザイン時に設定可能。実行時には意味を持たない。

イベント

表5は、Visual Basicアプリケーションで行われる操作と、それによって発生するイベントの一覧です。また表6は、イベントと、そのイベントを発生することができるオブジェクトをアルファベット順に示してあります。

表5　イベントと操作の対応

操作	イベント
コントロールに対する変更	Change, DropDown (コンボ ボックスのみ), PathChange, PatternChange (ファイル リスト ボックスのみ)
ドラッグ & ドロップ	DragDrop, DragOver
キー入力	KeyDown, KeyUp, KeyPress
マウスの操作	Click, DblClick, MouseDown, MouseUp, MouseMove
フォーカスの移動	GotFocus, LostFocus
タイマー間隔	Timer
フォームとピクチャー ボックスに関する操作	Paint, Resize, Load, Unload

表6　イベントとオブジェクトの対応

イベント＼オブジェクト	フォーム	ラベル	コマンドボタン	オプションボタン	チェックボックス	テキストボックス	リストボックス	コンボボックス	ピクチャーボックス	フレーム	タイマー	水平／垂直スクロールバー	ディレクトリリストボックス	ドライブリストボックス	ファイルリストボックス	メニュー	CLIPBOARD	PRINTER	SCREEN
Change		○				○		○				○	○	○					
Click	○	○	○	○	○		○	○	○				○		○	○			
DblClick	○	○		○			○	◎	○						○				
DragDrop	○	○	○	○	○	○	○	○	○	○		○	○	○	○				
DragOver	○	○	○	○	○	○	○	○	○	○		○	○	○	○				
DropDown								○											
GotFocus	○		○	○	○	○	○	○	○			○	○	○	○				
KeyDown	○		○	○	○	○	○	○	○			○	○	○	○				
KeyPress	○		○	○	○	○	○	○	○			○	○	○	○				
KeyUp	○		○	○	○	○	○	○	○			○	○	○	○				
Load	○																		
LostFocus	○		○	○	○	○	○	○	○			○	○	○	○				
MouseDown	○	○					○		○				○		○				
MouseMove	○	○					○		○				○		○				
MouseUp	○	○					○		○				○		○				
Paint	○								○										
PathChange															○				
PatternChange															○				
Resize	○																		
Timer											○								
UnLoad	○																		

◎　Style プロパティが1(シンプルコンボ)に設定されている場合のみ。

オブジェクト

表7は、プログラムやユーザーが行う可能性がある操作や状況と、それらの操作に対して良く使われる典型的なオブジェクトの一覧です。

表7　オブジェクトと操作の対応

操作	オブジェクト
画面全体に対する操作	SCREEN オブジェクト
リストから1項目を選択	リスト ボックス，コンボ ボックス， オプション ボタンのグループ
テキストのカット，コピー，ペースト	CLIPBOARD オブジェクト
ユーザーが変更できないテキストの表示	ラベル，ピクチャー ボックス
コントロールのグループ化	フレーム，ピクチャー ボックス
テキストの入力	テキスト ボックス
テキストやフォームの印刷	PRINTER オブジェクト
スクロール	水平または垂直スクロール バー
コマンドの選択	コマンド ボタン，メニュー
ファイルの選択	ファイル リスト ボックス
ディレクトリの選択	ディレクトリ リスト ボックス
ドライブの選択	ドライブ リスト ボックス
オプションの ON/OFF	チェック ボックス，オプション ボタン

第 2 部

A-Z リファレンス

オプション ボタン コントロール

機能

ユーザーが、いくつかの選択肢から 1 つを選択できるようにします。

解説

イベント

Click	DblClick	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus			

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

BackColor	Caption	CtlName	DragMode
Enabled	ForeColor	Height	Index
Left	MousePointer	Parent	TabIndex
TabStop	Tag	Top	Value
Visible	Width		

オプション ボタンは、オン/オフや True/False など、2 つの状態が切り換わるオプションを選択できるようにします。オプション ボタンはチェック ボックスと異なり、通常はグループ化して、ユーザーが複数の選択肢から 1 つを選択できるようにするために使われます。オプション ボタンは、同じコンテナ内に配置すると、自動的にグループ化され、その内の 1 つしか選択できないようになります。オプション ボタンはラジオ ボタンと呼ばれることもあります。

フレームやピクチャー ボックス内でオプション ボタンをグループ化するには、まずフレームまたはピクチャー ボックスをフォーム上に配置してから、その中にオプション ボタンをマウスでドラッグして配置して行きます。このようにすると、フレームまたはピクチャー ボックスはコンテナとして機能し、コンテナといっしょにその中のオプション ボタンも移動するようになります。1 度フォーム上に配置したオプション ボタンを、フレームやピクチャー ボックス内に移動しても、グループ化されませんから注意してください。

オプション ボタンとチェック ボックスの機能は、2 つの状態を切り換えるという意味では似ていますが、目的はまったく異なります。オプション ボタンは、グループ化して、複数の選択肢から 1 つを選択するために使います。これに対してチェック ボックスは、ある 1 つの項目のオン/オフを切り換えるために使います。

オプション ボタンの最小サイズは 1 x 3 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。

参照

チェック ボックス

コマンド ボタン コントロール

機能

ユーザーが選択すると、コマンドや動作が実行されるようにするためのコントロールです。

解説

イベント

Click	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

BackColor	Cancel	Caption	CtlName
Default	DragMode	Enabled	Height
Index	Left	MousePointer	Parent
TabIndex	TabStop	Tag	Top
Value	Visible	Width	

コマンド ボタンは、Visual Basic のダイアログ ボックス中によく現れる <了解> や <取消> ボタンのように、それを選択するとある動作が行われるようにしたいときに使います。

実行時にコマンド ボタンの Value プロパティに True (-1) を設定すると、そのコマンド ボタンの Click イベントが発生します。Value プロパティの値は、Click イベントに対応するイベント プロシージャ実行中には常に True (-1) で、終了すると False (0) にリセットされます。Value プロパティは、デザイン時には設定できません。

そのボタンがマウスでクリックされるか、そのボタンがフォーカスを持つときに [ENTER] または [SPACE] キーが押されると、Click イベントが発生します。例えばユーザーは、ダイアログ ボックスへの入力が完了したことを示すためにコマンド ボタンをクリックします。アプリケーションは、ダイアログ ボックスへのユーザー入力に基づいて処理を続行します。

コマンド ボタンにテキストを表示するには、Caption プロパティを設定してください。

ユーザーが [ESC] キーを押したときにイベントを発生させるには、Cancel プロパティを True (-1) に設定します。この設定は、1 つのフォーム中で 1 つのボタンだけに可能です。

[ENTER] キーを押したときに (フォーカスがないコマンド ボタンに対応する) イベントを発生させるには、Default プロパティを True (-1) に設定します。この設定は、1 つのフォーム中で 1 つのボタンだけに可能です。

コマンド ボタンの最小サイズは 1 x 3 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。Height に 1 または 2 を設定すると、このコントロールの外観は次のようになります。

コンボ ボックス コントロール

機能

テキスト ボックスとリスト ボックスを組み合せた機能で、ユーザーは値を直接入力することも、ドロップ ダウンするリストから選択することもできます。

解説

イベント

Change	Click	DblClick	DragDrop
DragOver	DropDown	GotFocus	KeyDown
KeyPress	KeyUp	LostFocus	

メソッド

ADDITEM	DRAG	MOVE	REFRESH
REMOVEITEM	SETFOCUS		

プロパティ

BackColor	CtlName	DragMode	Enabled
ForeColor	Height	Index	Left
List	ListCount	ListIndex	MousePointer
Parent	SelLength	SelStart	SelText
Sorted	Style	TabIndex	TabStop
Tag	Text	Top	Visible
Width			

コンボ ボックスは、テキスト ボックスとリスト ボックスの機能を組み合せたものです。ユーザーは、次のどちらの方法でも入力できます。

- テキストを直接入力する。
- ↓ をクリックして、ドロップ ダウンするリストから項目を選択する。

コンボ ボックスの編集部分とリスト部分の動作を指定するには、Style プロパティを使います。

ドロップ ダウンするリスト中の項目が選択されると、その文字列がテキスト ボックスに表示されます。実行時にコンボ ボックスのリスト部分の文字列を追加または削除するには、**ADDITEM** と **REMOVEITEM** メソッドを使ってください。

リスト ボックス部分に表示されている項目に実行時にアクセスするには、List, ListCount, ListIndex の各プロパティを使ってください。

テキスト ボックスとコンボ ボックスのテキスト ボックス部分には、あらかじめ Microsoft Windows に似たカット & ペーストの機能が用意されています。つまり、[CTRL]+[INS] または [CTRL]+[C] でクリップボードへのコピー (複写), [SHIFT]+[DEL] または [CTRL]+[X] でクリップボードへのカット (切り抜き), [SHIFT]+[INS] または [CTRL]+[V] でクリップボードからのペースト (貼り付け) が行えます。この機能は、1 つのテキスト ボックスまたはコンボ ボックス内だけでなく、複数のテキスト ボックスまたはコンボ ボックス間でも有効です。クリップボードの内容をコード中から操作したいときは、**CLIPBOARD** オブジェクトを使ってください。

コンボ ボックスの最小サイズは 1 x 5 で、Height と Width プロパティでサイズを設定します。Height プロパティは読み込み専用で、リスト部分の幅 (Width) だけしか変更できません。Width に 5 以下の正の整数を設定しても、5 になります。0 以下の値を設定するとエラーになります。

参照　リスト ボックス，テキスト ボックス

タイマー コントロール

機能　一定の時間が経過するたびにイベントを発生させます。

解説

イベント

Timer

プロパティ

CtlName	Enabled	Index	Interval
Left	Parent	Tag	Top

タイマー コントロールは、デザイン時にフォーム上に配置しますが、実行時には表示されません。Left および Top プロパティは、デザイン時に便宜的に表示されるだけで、実行時には意味を持ちません。

タイマー コントロールの Interval プロパティで設定した時間間隔ごとに、Timer イベントが発生します。一定時間ごとに実行したい処理を、Timer イベントに対するイベント プロシージャに記述します。

Timer イベントが発生するようにするには、Enabled プロパティを True (-1) に設定し、Interval プロパティを 1 以上の整数に設定します。実行時には、Enabled プロパティを False (0) に設定すれば、Timer イベントは発生しなくなります。

最大 16 個のタイマーを同時に利用できます。17 個目のタイマーを使おうとすると、ランタイム エラーが発生します。

参照　**TIMER** 関数，**TIMER** ステートメント

テキスト ボックス コントロール

機能

ユーザーからのテキスト入力を得られるようにします。アプリケーションからは、Text プロパティを使ってテキスト ボックスの内容を操作できます。

解説

イベント

Change	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

BackColor	BorderStyle	CtlName	DragMode
Enabled	ForeColor	Height	Index
Left	MousePointer	MultiLine	Parent
ScrollBars	SelLength	SelStart	SelText
TabIndex	TabStop	Tag	Text
Top	Visible	Width	

テキスト ボックスは、編集フィールドまたは編集コントロールと呼ばれることもあります。

テキスト ボックスは、最大 32KB のテキストを入力したり表示するために使います。テキスト ボックスの内容を、デザイン時にあらかじめ指定しておくこともできますし、実行時にユーザーがテキスト ボックスに入力することもできます。

テキスト ボックスとコンボ ボックスのテキスト ボックス部分には、あらかじめ Microsoft Windows に似たカット & ペーストの機能が用意されています。つまり、[CTRL]+[INS] または [CTRL]+[C] でクリップボードへのコピー (複写)，[SHIFT]+[DEL] または [CTRL]+[X] でクリップボードへのカット (切り抜き)，[SHIFT]+[INS] または [CTRL]+[V] でクリップボードからのペースト (貼り付け) が行えます。この機能は、1 つのテキスト ボックスまたはコンボ ボックス内だけでなく、複数のテキスト ボックスまたはコンボ ボックス間でも有効です。クリップボードの内容をコード中から操作したいときは、**CLIPBOARD** オブジェクトを使ってください。

デザイン時には、テキスト ボックスのサイズに関わらず、表示するデータは 1 行だけしか指定できません。実行時に複数行のテキストを表示するには、MultiLine プロパティを True (-1) に設定してください。

テキスト ボックスの最小サイズは 1 x 4 で、Height と Width の各プロパティの設定に対応します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。BorderStyle プロパティが 1 のときは、Height に 1 または 2 を設定すると次のような外観になります。

```
Height = 1              Height = 2              Height >= 3
```

テキスト ボックスには、水平および垂直のスクロール バーを付けることができます。ScrollBars プロパティを使い、デザイン時に設定します。必ず、MultiLine プロパティを True (-1) に設定してください。また、テキスト ボックスのサイズ (Width および Height プロパティ) がスクロール バーを表示できる最小サイズ以上でないと、スクロール バーは表示されません。

ディレクトリ リスト ボックス コントロール

機能

ディレクトリの一覧を表示し、ユーザーが実行時にディレクトリを選択できるようにするためのコントロールです。

解説

イベント

Change	Click	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

BackColor	CtlName	DragMode	Enabled
ForeColor	Height	Index	Left
List	ListCount	ListIndex	MousePointer
Parent	Path	TabIndex	TabStop
Tag	Top	Visible	Width

このコントロールは、ディレクトリの一覧を表示し、ユーザーがサブディレクトリを選択できるようにします。ファイルを操作するダイアログ ボックスを作成したいときなどに使います。

ディレクトリ名がディレクトリ リスト ボックスに表示される順番は、MS-DOS バージョン 5.0 で DIR /AD コマンドを実行した場合と同じです。

コード中から、ディレクトリ リスト ボックスに表示されている項目を調べたり変更したりするには、List, ListCount, ListIndex の各プロパティを使ってください。

選択されているディレクトリをファイル リスト ボックスに渡すには、Path プロパティを使います。例えば次のようにします。

```
SUB Dir1_Change ()
    File1.Path = Dir1.Path
END SUB
```

新しいドライブが選択されたときにディレクトリ リスト ボックスを更新するには、ドライブ リスト ボックスの Drive プロパティを使ってください。例えば次のようにします。

```
SUB Drive1_Change()
    Dir1.Path = Drive1.Drive
END SUB
```

ディレクトリ リスト ボックスの最小サイズは 4 x 5 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。

参照

ドライブ リスト ボックス，ファイル リスト ボックス

チェック ボックス コントロール

機能

オン/オフなど、2 つの状態を切り換えられるようにするためのコントロールです。チェック ボックス内に表示される X は、そのオプションが選択されていることを示します。

解説

イベント

Click	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

BackColor	Caption	CtlName	DragMode
Enabled	ForeColor	Height	Index
Left	MousePointer	Parent	TabIndex
TabStop	Tag	Top	Value
Visible	Width		

チェック ボックスは、オンまたはオフに設定できるオプションを表示します。[ ] の中に X が表示されていれば、設定がオンになっていることを示します。この状態を、チェックされていると呼びます。ユーザーは、チェック ボックスの状態を、次のいずれかの方法で切り換えられます。

- チェック ボックスをクリックする。
- チェック ボックスがフォーカスを持っているときにスペース キーを押す。

チェック ボックスのチェック/未チェックを切り換えたり、そのときの状態を得るには、Value プロパティを使います。

複数の選択肢をユーザーに示すために、チェック ボックスをまとめて使うことが可能です。同じ種類のコントロールは、コントロール配列を作成して 1 つのグループにまとめることができます。

チェック ボックスとオプション ボタンの機能は、2 つの状態を切り換えるという意味では似ていますが、目的はまったく異なります。オプション ボタンは、グループ化して、複数の選択肢から 1 つを選択するために使います。これに対してチェック ボックスは、ある 1 つの項目のオン/オフを切り換えるために使います。

チェック ボックスの最小サイズは 1 x 3 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。

参照　オプション ボタン

ドライブ リスト ボックス コントロール

機能　そのとき有効なドライブの一覧を表示し、ユーザーが実行時にドライブを選択できるようにするためのコントロールです。

解説

イベント

Change	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

BackColor	Ctlname	DragMode	Drive
Enabled	ForeColor	Height	Index
Left	List	ListCount	ListIndex
MousePointer	Parent	TabIndex	TabStop
Tag	Top	Visible	Width

このコントロールは、そのとき有効なフロッピー ディスク ドライブ、ハード ディスク、およびネットワーク上のドライブをアルファベット順に表示し、ユーザーがドライブを選択できるようにします。ファイルを操作するダイアログ ボックスを作成したいときなどに使います。

新しいドライブが選択されたときにディレクトリ リスト ボックスを更新するには、ドライブ リスト ボックスの Drive プロパティを使ってください。例えば次のようにします。

```
SUB Drive1_Change ()
    Dir1.Path = Drive1.Drive
END SUB
```

Path プロパティコード中から、ドライブ リスト ボックスに表示されている項目を調べたり変更したりするには、List, ListCount, ListIndex の各プロパティを使ってください。

ドライブ リスト ボックスの最小サイズは 1 x 5 で、Height と Width プロパティでサイズを設定します。Height プロパティは読み込み専用で、リスト部分の幅 (Width) だけしか変更できません。Width に 5 以下の正の整数を設定しても、5 になります。0 以下の値を設定するとエラーになります。

参照

ディレクトリ リスト ボックス，ファイル リスト ボックス

ピクチャー ボックス コントロール

機能

PRINT メソッドによって出力されたテキストを表示したり、別のコントロールをグループ化するために使います。

解説

イベント

Click	DblClick	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp
Paint			

メソッド

CLS	DRAG	MOVE	PRINT
REFRESH	SETFOCUS	TEXTHEIGHT	TEXTWIDTH

プロパティ

AutoRedraw	BackColor	BorderStyle	CtlName
CurrentX	CurrentY	DragMode	Enabled
ForeColor	Height	Index	Left
MousePointer	Parent	ScaleHeight	ScaleWidth
TabIndex	TabStop	Tag	Top
Visible	Width		

ピクチャー ボックスを使うと、罫線で囲んだ枠の中にテキストを表示することができます。このような枠のことを境界線またはボーダーと呼び、BorderStyle プロパティで制御します。なし，細線，太線が選べます。

ピクチャー ボックス中にテキストを表示するには、**PRINT** メソッドを使います。長いテキストを出力するとき、ピクチャー ボックスの枠を超える部分は表示されませんから注意してください。

ピクチャー ボックスは、フォーム上のコントロールをグループ化するためのコンテナとしても使えます。例えば、あるグループのオプション ボタンを別のグループと分けて表示すれば、フォームを見やすくすることができます。

コントロールをグループ化するには、ピクチャー ボックスを配置してから、その中にコントロールを置きます。このようにすると、ピクチャー ボックスはコンテナとして機能し、ピクチャー ボックスといっしょにその中のコントロールも移動するようになります。

1 つのピクチャー ボックス中には、最大 253 個のコントロールを配置できます。

ピクチャー ボックスは、その中にコントロールが配置されているときでも、フォーカスを持つことができます。

ピクチャー ボックスの内容を常に最新の状態で表示しておくには、AutoRedraw プロパティを True (-1) に設定するか、Paint イベント プロシージャ内で **PRINT** メソッドを使ってください。

ピクチャー ボックスの最小サイズは 1 x 1 で、Height と Width プロパティでサイズを設定します。0 以下の値を設定するとエラーになります。

参照　フレーム，オプション ボタン

ファイル リスト ボックス コントロール

機能　ディレクトリ中のファイルの一覧を表示し、ユーザーが実行時にファイルを選択できるようにするためのコントロールです。

解説

イベント

Click	DblClick	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp
PathChange	PatternChange		

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

Archive	BackColor	CtlName	DragMode
Enabled	FileName	ForeColor	Height
Hidden	Index	Left	List
ListCount	ListIndex	MousePointer	Normal
Parent	Path	Pattern	ReadOnly
System	TabIndex	TabStop	Tag
Top	Visible	Width	

このコントロールは、指定されたディレクトリ中のファイルをアルファベット順に表示し、ユーザーがファイルを選択できるようにします。ファイルを操作するダイアログ ボックスを作成したいときなどに使います。Path および Pattern プロパティで特に指定しなければ、カレント ディレクトリ中のすべてのファイルが一覧表示されます。

ファイル リスト ボックスに対して発生した Click イベントは、MouseDown, MouseUp, Click の順番で認識されます。また DblClick イベントは、MouseDown, MouseUp, Click, DblClick の順番で認識されます。

表示されるファイルの種類は、Hidden, Archive, ReadOnly, System, Normal の各プロパティで決定します。

コード中から、ファイル リスト ボックスに表示されている項目を調べたり変更したりするには、List, ListCount, ListIndex の各プロパティを使ってください。

ファイル リスト ボックスの最小サイズは 4 x 5 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。

参照

ディレクトリ リスト ボックス, ドライブ リスト ボックス

フォーム

機能

フォームは、アプリケーションのユーザー インターフェイスの基本となるウィンドウです。フォームの上にコントロールを配置して行きます。つまりコントロールのコンテナとして機能します。

解説

イベント

Click	DblClick	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
Load	LostFocus	MouseDown	MouseMove
MouseUp	Paint	Resize	UnLoad

メソッド

CLS	DRAG	HIDE	MOVE
PRINT	PRINTFORM	REFRESH	SHOW
SETFOCUS	TEXTHEIGHT	TEXTWIDTH	

プロパティ

AutoRedraw	BackColor	BorderStyle	Caption
ControlBox	CurrentX	CurrentY	Enabled
ForeColor	FormName	FormType	Height
Left	MaxButton	MinButton	MousePointer
Parent	ScaleHeight	ScaleWidth	Tag
Top	Width	WindowState	Visible

フォームもコントロールと同じようにオブジェクトの一種であり、対応するイベント，適用可能なメソッド，持っているプロパティが決まっています。同時に複数のフォームを画面に表示することが可能であり、それぞれに対して外観やイベント発生時の応答を指定して行きます。

フォーム モジュールを作成するには、[ﾌｧｲﾙ] メニューから [新規フォーム] コマンドを選択します。

フォーム モジュールに含まれているプロシージャは、たとえそれがジェネラル プロシージャであっても、別のモジュールから呼び出すことはできません。これに対してコード モジュールに含まれているプロシージャは、別のモジュールからも呼び出すことができます。

フォーム モジュールのモジュール レベルに、実行可能ステートメントを置くことはできません。

通常、1 つのアプリケーションのユーザー インターフェースを設計するには、複数のフォームを使います。

1 つのフォームには最大 254 個のコントロールを配置できます。フォームに配置したコントロールは、そのフォームの子 (child) として参照されます。フォーム上のコントロールのいずれかがフォーカスを持つことができれば、そのフォーム自体はフォーカスを持つことができません。

フォームの大きさは、Height と Width プロパティで設定します。

コントロールを持たないデフォルトのサイズのフォームは、約 96 バイトのメモリを消費します。ユーザーは **FRE** 関数を使って、フォームに配置したコントロールに使われているおよそのメモリ量を知ることができます。例を次に示します。

```
SUB Form_Click ()
    Loaded! = FRE(-1)
    UNLOAD Form1
    Unloaded! = FRE(-1)
    SHOW
    PRINT STR$(Unloaded! - Loaded!)
END SUB
```

参照　**$FORM** メタコマンド

フレーム コントロール

機能

オプション ボタンなどの他のコントロールをグループ化するための、四角形の領域を表示します。

解説

イベント

DragDrop	DragOver

メソッド

DRAG	MOVE	REFRESH

プロパティ

BackColor	Caption	CtlName	DragMode
Enabled	ForeColor	Height	Index
Left	MousePointer	Parent	TabIndex
Tag	Top	Visible	Width

フレームは、フォーム上のコントロールをグループ化するためのコンテナとして使います。例えば、あるグループのチェック ボックスを別のグループと分けて表示すれば、フォームを見やすくすることができます。また、オプション ボタンをグループ化するためによく使います。

コントロールをグループ化するには、フレームを配置してから、その中にコントロールを置きます。このようにすると、フレームはコンテナとして機能し、フレームといっしょにその中のコントロールも移動するようになります。フォーム上の別の場所に生成したコントロールをフレームの中に移動しても、グループ化されませんから注意してください。

1 つのフレーム中には、最大 253 個のコントロールを配置できます。

コントロールを含むフレーム全体に対し、タイトルとしてヘッダーを付けるには、Caption プロパティを使います。

フレームの最小サイズは 1 x 1 で、Height と Width プロパティでサイズを設定します。0 以下の値を設定するとエラーになります。フレームを水平または垂直のセパレータとして使うには、Height または Width プロパティを 1 に設定してください。例えば Height を 1 に設定すると、フレームは次のように表示されます。

```
Height = 1                Height >= 2
```

参照

ピクチャー ボックス

メニュー コントロール

機能

メニューを表示し、ユーザーがコマンドを選択できるようにします。メニューのタイトルやコマンドそれぞれが 1 つのメニュー コントロールです。

解説

イベント

Click

プロパティ

Caption	Checked	CtlName	Enabled
Index	Parent	Separator	Tag
Visible			

メニュー コントロールを使うと、アプリケーションに独自のメニューを加えることができます。メニューは、メニュー タイトル，メニュー コマンド，セパレータ バーから構成されます。ドロップ ダウンするメニュー中に表示される各項目を、メニュー項目と呼びます。

ユーザーがメニュー タイトルを選択すると、メニューがドロップ ダウンします。例えば、[ﾌｧｲﾙ] メニューを選択したら、[新規作成]，[オープン]，[終了] などのメニュー項目が表示されるようにします。

メニューの作成は、フォーム デザイナのメニュー デザイン ウィンドウを使って行います。

メニュー タイトルや各メニュー項目は、それぞれ 1 つのメニュー コントロールです。それぞれのメニュー コントロールに別々の名前を付けることもできますし、コントロール配列を作成してグループ化することもできます。実行時にコントロール配列にメニュー項目を追加するには、**LOAD** ステートメントを使います。

フォームの上部に表示され、メニュー バーを構成するメニュー コントロールを、メニュー タイトルと呼びます。また、ドロップ ダウンするメニュー中に表示されるメニュー コントロールを、メニュー項目と呼びます。

メニューは、5 段階までの階層を持つことができます。子メニューがドロップ ダウンする項目を、“トップ レベル” にあると言います。メニュー タイトルはつねにトップ レベルです。

有効な (Enabled プロパティが True に設定されている) メニュー タイトルまたはメニュー項目が選択されると、Click イベントが発生します。また、選択された項目がトップ レベルなら、特にイベント プロシージャを記述しなくても、自動的にその下の階層のメニューがドロップ ダウンします。トップ レベルの項目で発生した Click イベントに対応するイベント プロシージャを記述すれば、そのときの状態によって、表示するメニューの内容を変えることが可能です。

メニュー コントロールは、メニュー項目の外観と機能を制御するいくつかのプロパティを持っています。メニュー コントロールのプロパティの機能は次のとおりです。

プロパティ	設定	説明
Caption	- (マイナス)	その項目はセパレータ バーとして表示されます。
Caption	& (文字の前)	次の文字をアクセス キーに設定します。[メニュー] キーを押しながらその文字を押すと、そのメニュー項目が選択されます。
Checked	True (-1)	その項目の左にチェック マーク (*) が付きます。トップ レベルの項目に対して指定することはできません。
Enabled	True (-1)	その項目はイベントを認識します。選択されると、コマンドを実行するか、次のレベルのメニューを表示します。
Enabled	False (0)	その項目はイベントを認識しません。選択されても、何の動作も行いません。
Visible	False (0)	その項目はメニューに表示されません。その後のメニュー項目が繰りあがって表示されます。
Separator	True (-1)	その項目はセパレータ バーとして表示されます。

メニュー項目を区切るためにセパレータ バーを表示するには、Caption プロパティを "-" (マイナスのキャラクタ) に設定するか、Separator プロパティを True (-1) に設定します。メニュー デザイン ウィンドウで [セパレータ] チェック ボックスを選択すると、Separator プロパティは True (-1) に設定されます。セパレータ バーと指定されている項目は、Click イベントを認識しません。

メニュー デザイン ウィンドウの [チェック] チェック ボックスを選択すると、Checked プロパティが True (-1) に設定され、メニュー項目の左にチェック マーク (*) が表示されます。チェック マークは、その項目がチェックされているということをユーザーに示すだけで、プログラム中でどんな意味にも使うことができます。トップ レベルの項目にチェック マークを付けることはできません。

ラベル コントロール

機能

ユーザーが直接変更できない (キャプションと呼ばれる) テキストを表示します。

解説

イベント

Change	Click	DblClick	DragDrop
DragOver	MouseDown	MouseMove	MouseUp

メソッド

DRAG	MOVE	REFRESH

プロパティ

Alignment	AutoSize	BackColor	BorderStyle
Caption	CtlName	DragMode	Enabled
ForeColor	Height	Index	Left
MousePointer	Parent	TabIndex	Tag
Top	Visible	Width	

イベントに応答して、ラベルの内容を変えて表示すれば、ユーザーへのメッセージとしてラベルを利用できます。例えば、長い時間がかかる処理の場合、進行状況を示すメッセージを表示できます。

ラベルは、Caption プロパティを持たないコントロール (テキスト ボックスなど) に対して、説明のテキストを表示するためにも使えます。もちろん、Caption プロパティを持つコントロールに対して利用してもかまいません。

ラベルの最小サイズは 1 x 1 で、Height と Width プロパティでサイズを設定します。0 以下の値を設定するとエラーになります。BorderStyle プロパティでラベルに境界線を設定しているとき、Height プロパティに 1 または 2 を設定すると、次のような外観になります。

```
Height = 1              Height = 2              Height >= 3
```

参照

Caption プロパティ

リスト ボックス コントロール

機能

選択肢の一覧を表示し、ユーザーがその中の 1 つを選択できるようにします。

解説

イベント

Click	DblClick	DragDrop	DragOver
GotFocus	KeyDown	KeyPress	KeyUp
LostFocus	MouseDown	MouseMove	MouseUp

メソッド

ADDITEM	DRAG	MOVE	REFRESH
REMOVEITEM	SETFOCUS		

プロパティ

BackColor	CtlName	DragMode	Enabled
ForeColor	Height	Index	Left
List	ListCount	ListIndex	MousePointer
Parent	Sorted	TabIndex	TabStop
Tag	Text	Top	Visible
Width			

Sorted プロパティに True (-1) を設定すると、リストはアルファベット順に表示されます。

リスト中の項目は、**ADDITEM** メソッドを使って追加、**REMOVEITEM** メソッドを使って削除できます。

コード中から、リスト ボックスに表示されている項目を調べたり変更したりするには、List, ListCount, ListIndex の各プロパティを使ってください。

リスト ボックスの最小サイズは 4 x 5 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。

参照

コンボ ボックス

水平，垂直スクロール バー コントロール

機能

水平または垂直方向のスクロール バーを表示し、ある範囲内での位置を表示および設定します。

解説

イベント

Change	DragDrop	DragOver	GotFocus
KeyDown	KeyPress	KeyUp	LostFocus

メソッド

DRAG	MOVE	REFRESH	SETFOCUS

プロパティ

Attached	CtlName	DragMode	Enabled
Height	Index	LargeChange	Left
Max	Min	MousePointer	Parent
SmallChange	TabIndex	TabStop	Tag
Top	Value	Visible	Width

スクロール バーは、画面をスクロールさせ、長いリストや大量の情報を素早く操作したり、その範囲内での現在位置を示すために使います。このようにフォームの全体をスクロールするために使うときは、デザイン時に Attached プロパティを True (-1) に設定しておきます。

スクロール バーは、スピードや量のインジケータとしても使えます。例えば、コンピュータ ゲームの音量を調節したり、時間を計っている処理で経過時間を示したりすることに利用できます。この場合、デザイン時に Attached プロパティを False (0) に設定しておきます。また、Max および Min プロパティで、適切な範囲を設定してください。

スクロール バーのバーの部分がクリックされたときの変更量を指定するには LargeChange プロパティを使い、バーの両端にある矢印がクリックされたときの変更量を指定するには SmallChange プロパティを使います。そのスクロール バーの Value プロパティの値は、LargeChange プロパティと SmallChange プロパティに設定されている値を単位として増減します。

Value プロパティに 0～32,767 の値を設定することで、実行時にスクロール バー中の位置を設定できます。

水平スクロール バーと垂直スクロール バーは、方向が異なっているだけで、機能は同じです。

水平スクロール バー コントロールの最小サイズは 1 x 2，垂直スクロール バー コントロールの最小サイズは 2 x 1 で、Height と Width プロパティでサイズを設定します。これ以下の正の整数を設定しても、最小値になります。0 以下の値を設定するとエラーになります。

参照

ScrollBars プロパティ

ABS 関数

機能　数式の絶対値を返す数値演算関数です。

構文　**ABS**(*numeric-expression*)

numeric-expression　任意の数式。絶対値とは、引数として指定された数値の正負の記号を除いた数値そのものです。たとえば、 ABS(-2) と ABS(2) の返す値は、どちらも 2 です。

参照　**SGN** 関数

Absolute ルーチン

機能　マシン語のプロシージャに制御を渡すルーチンです。

構文　**CALL Absolute** ([*argumentlist*,]*offset%*)

argumentlist　プロシージャに引き渡す引数。プロシージャには、その引数のカレント データ セグメントからのオフセットが渡されます。

offset%　**DEF SEG** ステートメントで設定したカレントのコード セグメントの先頭から、プロシージャの開始位置までのオフセット値。

解説　このルーチンは、VBDOS.EXE 本体ではなく、Quick ライブラリ VBDOS.QLB およびオブジェクト モジュール ライブラリ VBDOS.LIB に含まれています。VBDOS.EXE 起動時に、/L オプションを指定してください。また、VBDOS.BI をインクルードしてください。

Absolute ルーチンは、以前のバージョンの QuickBasic, Microsoft Basic との互換性を保つために残されています。**CALL** ステートメントと **DECLARE** ステートメントを使うと、より容易にアセンブリ言語と Basic の複数言語によるプログラミングができます。

引数はオフセットとして引き渡されますが、マシン語のプログラムは far 呼び出しで呼び出されます。

offset% はプロシージャには渡されません。Absolute ルーチンを実行する前に、**DEF SEG** ステートメントを使って、呼び出すルーチンのコード セグメントを設定しなければなりません。

引数 *offset%* に整数でない値を指定すると、エラーになります。

参照　**CALL** ステートメント (Basic プロシージャ), **DEF SEG** ステートメント, **$INCLUDE** メタコマンド, Interrupt[X] ルーチン, **VARPTR** 関数, **VARSEG** 関数

ActiveControl, ActiveForm プロパティ

機能

ActiveControl プロパティは、そのときアクティブなコントロールを示すハンドルを返します。ActiveForm プロパティは、そのときアクティブなフォームを示すハンドルを返します。どちらもデザイン時には使用できず、実行時には読み込み専用です。

構文

SCREEN.ActiveControl

SCREEN.ActiveForm

対象

SCREEN オブジェクト

解説

ActiveControl プロパティは、そのときフォーカスを持っているコントロールを示すハンドルを返します。また ActiveForm プロパティは、そのときフォーカスを持っているコントロールを含むフォーム、またはフォーカスを持っているフォームを示すハンドルを返します。どちらのプロパティも、値を設定することはできません。また、デザイン時には利用できません。コード中からフォーカスを移動するには、**SETFOCUS** メソッドを使います。

フォーム上のすべてのコントロールが非表示または無効になっている場合、およびフォーム上に 1 つもコントロールが配置されていない場合は、ActiveControl プロパティを参照するとエラーが発生します。また、タイマーやラベルなどのコントロールはフォーカスを持てないことに注意してください。

フォームが 1 つもロードされていない場合、ActiveForm プロパティを参照するとエラーになります。

これらのプロパティは、そのときフォーカスを持っているフォームやコントロールをコード中で指定するために使います。次のように、プロパティやメソッドも指定できます。

```
SCREEN.ActiveControl.Tag = "OkButton"
```

フォーカスを持っているコントロールの種類を調べたいときは、**IF** ステートメントと **TYPEOF** キーワードを使います。また、Tag プロパティをあらかじめ設定しておくと、同じ種類のコントロールが複数ある場合でも、1 つのコントロールを特定することができます。

参照

GotFocus イベント，LostFocus イベント

ADDITEM メソッド

機能

実行時にリスト ボックスまたはコンボ ボックスに項目を追加します。

構文

[*formname*.]*ctlname*.**ADDITEM** *item$*[, *index%*]

ctlname	リスト ボックスまたはコンボ ボックスの名前。
item$	リストに加える文字列を示す文字列式。
index%	リスト中で *item$* が挿入される位置を指定する整数。0 が先頭を示します。最大値は ListCount プロパティの値 -1 になります。

対象

コンボ ボックス，リスト ボックス

解説

index% が省略されたとき、Sorted プロパティが True (-1) に設定されていれば、ソートされた位置に *item$* が加えられます。False (0) であれば、リストの最後に *item$* が加えられます。

Sorted プロパティは大文字小文字を区別しません。大文字小文字を区別してソートしたいときは、そのためのルーチンを作成してください。Sorted プロパティが True (-1) に設定されているとき、**ADDITEM** で *index%* を指定しないように注意してください。*index%* にソートされた正しい位置を指定しないと、それ以降に追加される要素は正しくソートされなくなります。

リスト ボックス部分に表示されている項目に実行時にアクセスするには、List, ListCount, ListIndex の各プロパティを使ってください。

リスト ボックス部分の項目の内容を変更するときは、**ADDITEM** メソッドではなく List プロパティを使います。

```
Combo1.List(1) = "Students"
```

参照

REMOVEITEM メソッド

Alignment プロパティ

機能　ラベル中のテキストの表示方法を決めます。

構文　[*formname*.]*ctlname*.**Alignment**[= *setting%*]

ctlname	ラベルの名前。
setting%	表示方法を示す整数。0 が左詰め，1 が右詰め，2 が中央揃えを示します。デフォルトは 0 (左詰め) です。

対象　ラベル

参照　AutoSize プロパティ

Archive, Hidden, Normal, ReadOnly, System プロパティ

機能　ファイル リスト ボックスに、どの属性のファイルを表示するかを決めます。

構文　[*formname*.]*ctlname*.**Archive**[= *boolean%*]

[*formname*.]*ctlname*.**Hidden**[= *boolean%*]

[*formname*.]*ctlname*.**Normal**[= *boolean%*]

[*formname*.]*ctlname*.**ReadOnly**[= *boolean%*]

[*formname*.]*ctlname*.**System**[= *boolean%*]

ctlname	ファイル リスト ボックスの名前。
boolean%	その属性を持つファイルを表示するかどうかを示す値。 True (-1) は表示することを示し、 False (0) は表示しないことを示します。

対象　ファイル リスト ボックス

解説　これらのプロパティは、ファイル リスト ボックスに表示するファイルの種類を規定しています。ファイルの属性はオペレーティング システムが管理しており、Visual Basic for MS-DOS では Archive (アーカイブ)，Hidden (隠しファイル)，Normal (通常)，ReadOnly (読み込み専用)，System (システム ファイル) の 5 つの属性が利用可能です。Normal は、System および Hidden 属性を持たないファイルを示します。Normal, System, Hidden の各プロパティを True (-1) に設定すると、すべてのファイルが表示されます。System と Hidden の両方の属性を持つファイルを表示するには、System と Hidden の両方のプロパティを True (-1) に設定する必要があります。

これらのプロパティのデフォルトの設定は、Archive, Normal, ReadOnly が True (-1)、Hidden, System が False (0) です。

参照　Path プロパティ，Pattern プロパティ，**SYSTEM** ステートメント

ASC, CHR$ 関数

機能　**ASC** 関数は、引数として指定した文字列の、最初の文字の ASCII コードまたは漢字連続コードを返す、文字列処理関数です。**CHR$** 関数は、指定した ASCII コードまたは漢字連続コードに対応する文字を返す、文字列処理関数です。

構文

ASC(*string-expression$*)

CHR$(*code1%*[,*code2%*[,...]])

string-expression$	変換する文字列を示す文字列式。
coden%	ASCII 文字コードまたは漢字連続コードを示す、 0～11535 の値を持つ数式。

解説

ASC 関数と **CHR$** 関数は、互いにほぼ逆の機能を持っています。**CHR$** 関数は数値を 1 文字の文字列に変換します。**ASC** 関数は文字列を数値に変換します。

string-expression$ に null 文字を指定すると、実行時に "引数が有効範囲にありません" エラーが発生します。

CHR$** 関数は、通常、画面やプリンタへ特殊文字を送るときに使います。例えば、改行を示すコードは、CHR$(&HD) + CHR$(&HA) と表すことができます。また **CHR$ 関数は、次のように文字列にダブルクォーテーション (") を付けるときにも使います。

```
Msg$ = CHR$(34) + "Quoted string" + CHR$(34)
```

この文字列を出力すると、画面には次のように表示されます。

```
"Quoted string"
```

参照　**KTN$** 関数，**JIS$** 関数

ATN 関数

機能

引数に指定した数値のアークタンジェントを返す数値演算関数です。アークタンジェントとは、タンジェントの逆関数です。

構文

ATN(*numeric-expression*)

numeric-expression 直角をはさむ両辺の適切な割合を示す数式。

解説

ATN (アークタンジェント) はタンジェント (**TAN**) の逆関数で、直角をはさむの 2 辺の割合を引数に取り、対応する角度を返します。割合とは、求める角の反対の辺の長さと、その角に隣接する辺の割合です。

戻り値として、$-\pi/2$ ～ $\pi/2$ の値 (単位はラジアン) を返します。ここで、π は約 3.141593, $\pi/2$ ラジアン = 90 度です。

numeric-expression は、いずれの数値データ型でもかまいません。*numeric-expression* が整数または単精度の値であると単精度で計算します。その他の数値データ型であると、倍精度で計算します。

参照

COS 関数, **SIN** 関数, **TAN** 関数

Attached プロパティ

機能

スクロール バーがフォームの下端または右端に設定されるかどうかを決めます。実行時には読み込み専用です。

構文

[*formname*.]*ctlname*.**Attached**

ctlname 水平または垂直スクロール バーの名前。

対象

水平スクロール バー, 垂直スクロール バー

解説

Attached プロパティは、デザイン時にフォーム デザイナで True (-1) または False (0) に設定できます。実行時には変更できません。デフォルトは False (0) です。

Attached プロパティは、水平または垂直スクロール バーが、フォームの下端または右端の適切な位置に設定されているかどうかを示します。False (0) に設定されている場合、スクロール バーはフォーム上に配置されているだけで、フォームのサイズを変更してもスクロール バーのサイズは同じです。スクロール バーをスピードや量のインジケータとして使う場合にこの設定にします。この設定がデフォルトです。True (-1) に設定されている場合、スクロール バーはフォームの適切な位置に配置されており、フォームのサイズを変更すると、自動的にスクロール バーのサイズも変更されます。

Attached プロパティが True (-1) に設定されているとき、実行時に Left, Height, Width, Top プロパティを変更しようとするとエラーが発生します。

フォームに対して Attached プロパティが True (-1) の状態で同時に機能するのは、水平および垂直スクロール バーそれぞれ 1 つだけです。

フォームに境界線があるときに、Attached プロパティが True (-1) の状態でスクロール バーを配置すると、その部分の境界線は表示されません。

スクロール バーの色は、Attached プロパティの設定に影響を受けます。Attached プロパティが False (0) のスクロール バーの色は、**SCREEN** オブジェクトの ControlPanel プロパティの SCROLLBAR_BACKCOLOR と SCROLLBAR_FORECOLOR 要素の設定によって決まります。Attached プロパティが True (-1) のスクロール バーの色は、**SCREEN** オブジェクトの ControlPanel プロパティの ACTIVE_BORDER_BACKCOLOR および ACTIVE_BORDER_FORECOLOR 要素の設定によって決まります。

参照

Scrollbars プロパティ

AutoRedraw プロパティ

機能

フォームまたはピクチャー ボックスを、必要に応じて自動的に再描画するかどうかを決めます。

構文

[*formname.*][*ctlname.*]**AutoRedraw**[= *boolean%*]

ctlname	ピクチャー ボックスの名前。
boolean%	自動的に再描画するかどうかを示す値。 True (-1) なら行い、False (0) なら行いません。デフォルトは False (0) です。

対象

フォーム，ピクチャー ボックス

解説

AutoRedraw プロパティは、フォームやピクチャー ボックスに対して **PRINT** メソッドで出力されたテキストを、必要に応じて自動的に再描画するかどうかを示します。True (-1) に設定されていると、**PRINT** メソッドによる出力はメモリ中にも保持され、オブジェクトのサイズを変更したときや、他のフォームに隠されていたオブジェクトが現れたときに、メモリ中のデータを使って自動的に再描画されます。

AutoRedraw プロパティが True (-1) に設定されていると、Paint イベントは発生しません。

AutoRedraw プロパティの設定にかかわらず、フォームやピクチャー ボックスの BackColor プロパティを変更すると、すべての出力は消去されます。ただし、次の出力位置は変わりません。

フォームやピクチャー ボックスに、AutoRedraw プロパティが True (-1) に設定されている間に表示されたテキストは、**CLS** メソッドを使う前に AutoRedraw プロパティを False (0) に変更すると、クリアされません。ただしこの場合でも、次の出力位置はフォームやピクチャー ボックスの左上に再設定されますから注意してください。また、AutoRedraw プロパティを再び True (-1) に戻してから **CLS** メソッドを使えばクリアされます。AutoRedraw プロパティを細かく操作すれば、クリアされるものとされないものを出力時に区別できます。

再描画の必要がないフォームやピクチャー ボックスでは、使用するメモリを節約するために、AutoRedraw プロパティは False (0) に設定しておいてください。

参照　AutoSize プロパティ，Paint イベント，**REFRESH** メソッド

AutoSize プロパティ

機能　ラベルのサイズを、表示されるテキストの長さに合わせて自動的に調整するかどうかを決めます。

構文　[*formname.*]*ctlname.***AutoSize**[= *boolean%*]

boolean%　自動的に調整するかどうかを示す値。True (-1) なら調整し、False (0) なら調整しません。デフォルトは False (0) です。

対象　ラベル

解説　False (0) に設定されているとき、長いテキストを表示しようとすると、そのときラベルコントロールで表示できる長さを超える部分は表示されません。

参照　Alignment プロパティ，AutoRedraw プロパティ

BackColor, ForeColor プロパティ

機能

オブジェクトの背景色 (BackColor) と文字色 (ForeColor) を決めます。

構文

[*formname.*][*ctlname.*]**BackColor**[= *color%*]

[*formname.*][*ctlname.*]**ForeColor**[= *color%*]

color% オブジェクトの背景色および文字色を示す、0～15 の色番号。BackColor プロパティのデフォルトは 7 (白)，ForeColor プロパティのデフォルトは 0 (黒) です。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス

解説

フォームの背景色を変更すると、フォームのバックグラウンドの再描画が行われ、AutoRedraw が True (-1) に設定されていても、バックグラウンド中のすべての文字がクリアされてしまいます。

色を示す定数は、インクルード ファイル CONSTANT.BI で定義されています。

色番号は 0～15 の範囲ですが、実際に表示可能な色数はハードウェアに依存します。

コマンド ボタンには ForeColor プロパティがありません。文字色は、SCREEN オブジェクトの ControlPanel プロパティで設定します。

参照

ControlPanel プロパティ

BEEP ステートメント

機能

スピーカーからビープ音を発生させます。

構文

BEEP

解説

BEEP ステートメントは、PRINT CHR$(7) を実行するのと同じです。

参照

PLAY ステートメント (ミュージック)，**PRINT** ステートメント，**SOUND** ステートメント

BLOAD, BSAVE ステートメント

機能

BSAVE ステートメントは、あるメモリ領域の内容を、ファイルまたはデバイスに転送します。**BLOAD** ステートメントは、**BSAVE** ステートメントで作成したメモリイメージ ファイルを、入力ファイルまたはデバイスからメモリに読み込みます。

構文

BSAVE *filespec$, offset%, length%*

BLOAD *filespec$*[, *offset%*]

filespec$	メモリイメージ ファイルを書き込むまたは読み出すファイルまたはデバイスを示す文字列式。
offset%	メモリ領域の先頭アドレスを示す数式。
length%	保存するメモリ領域のバイト数を示す、0～65,535 の符号なし整数。

解説

BSAVE ステートメントと **BLOAD** ステートメントを使うと、画面イメージをファイルに保存し、後でそのイメージを復元することができます。ただしこの操作は、画面に表示されているすべてのフォームに影響を与える可能性があります。フォームが表示されているときに **BSAVE** や **BLOAD** ステートメントを使うには、表示されているすべてのフォームをアンロードするか非表示にする必要があります。

BSAVE ステートメントを使うと、データやプログラムをメモリイメージ ファイルとしてディスクに保存できます。メモリイメージ ファイルとは、メモリに格納されていた情報をバイト単位でコピーしたものと、そのファイルをロードするときに **BLOAD** ステートメントが利用する情報から成ります。

BLOAD ステートメントを使うと、メモリイメージ ファイルとして保存されているプログラムやデータをメモリに読み込むことができます。

読み込み開始アドレスは、*offset%* に指定した値と、直前に実行された **DEF SEG** ステートメントで決まります。*offset%* を省略すると、ファイルに含まれているセグメント アドレスとオフセット アドレス (**BSAVE** ステートメントで使ったアドレス) が使われます。つまりファイルは、保存したときに使ったアドレスで読み込まれます。

offset% を指定すると、使われるセグメント アドレスは、直前に実行された **DEF SEG** ステートメントで設定されたセグメントになります。**DEF SEG** ステートメントが実行されていないと、Visual Basic のデータセグメント (DGROUP) と見なされます。

offset% に長整数，単精度または倍精度の数値を指定すると、整数に変換されます。*offset%* が -1～-32,768 の範囲の負の数であると、符号なしの 2 バイト オフセットとして扱われます。

BLOAD ステートメントを使って、ファイルを拡張 (EMS) メモリ配列に読み込まないでください。また、**BSAVE** ステートメントを使って、拡張 (EMS) メモリ配列を出力ファイルに転送しないでください。

/Ea オプションを指定して Visual Basic (VBDOS.EXE) を起動した場合、次の配列は EMS メモリに格納されることがあります。

- 512 バイト～ 16K バイトの数値配列
- 512 バイト～ 16K バイトの固定長文字列配列
- 512 バイト～ 16K バイトのユーザー定義型配列

16K バイト以下の配列に対して **BLOAD** または **BSAVE** ステートメントを使うときは、/Ea オプションを指定せずに Visual Basic (VBDOS.EXE) を起動してください。この場合、配列は EMS メモリには格納されません。

BLOAD ステートメントはアドレスの範囲のチェックをしないので、メモリ上のどこにでもファイルを読み込むことができます。このため、Visual Basic やオペレーティングシステムが常駐している領域にファイルを上書きしてしまわないように注意する必要があります。

スクリーン モードによってメモリの使い方が異なるため、グラフィック イメージをロードするときは、それを作成したときと同じスクリーン モードで行うようにしてください。

参照　**DEF SEG** ステートメント

BorderStyle プロパティ

機能　フォームやコントロールの境界のスタイル (枠線) を決めます。

構文　[*formname.*][*ctlname.*]**BorderStyle**[= *setting%*]

setting%　境界のスタイルを示す整数。

対象　フォーム，ラベル，ピクチャー ボックス，テキスト ボックス

解説　フォームおよびテキスト ボックスの BorderStyle プロパティは、実行時には読み込み専用です。ラベルとピクチャー ボックスでは実行時にも取得，設定ができます。

フォームの BorderStyle プロパティに設定可能な値は次のとおりです。

値	意味	機能
0	なし	タイトル バー，最小化/最大化ボタン，コントロールメニュー ボックス，Caption プロパティの値は表示されない。サイズ固定。
1	細線 (サイズ固定)	サイズ変更は最小化 /最大化のみ。
2	細線 (サイズ可変)	(デフォルト) サイズ変更可能。
3	太線 (サイズ固定)	サイズ変更は最小化 /最大化のみ。
4	太線 (サイズ可変)	サイズ変更可能。
5	極太 (サイズ固定)	サイズ変更は最小化 /最大化のみ。
6	極太 (サイズ可変)	サイズ変更可能。

キャプション (Caption プロパティの値)，コントロール メニュー ボックス，最小化/最大化ボタンが表示されるかどうかは、それぞれ別のプロパティによって設定します。ただし、BorderStyle プロパティが 0 に設定されている場合は、これらは表示されません。

サイズ固定の設定では、フォームの四隅をドラッグしてもサイズは変更されず、コントロール メニューのサイズ変更コマンドも無効になります。ただし、最大化/最小化は可能です。

フォームにタイトル バーが表示されていない場合は、実行時にユーザーがフォームを移動することはできません。

フォームの BorderStyle プロパティは、フォームを使ってダイアログ ボックスを実現するときに便利です。通常、ダイアログ ボックスを作成するときは、BorderStyle プロパティを 3 (太線サイズ固定) にします。

フォームの境界の色は、**SCREEN** オブジェクトの ControlPanel プロパティの ACTIVE_BORDER_BACKCOLOR および ACTIVE_BORDER_FORECOLOR 要素の設定によって決まります。

ラベル，ピクチャー ボックス，テキスト ボックスの BorderStyle プロパティに設定可能な値は次のとおりです。

値	意味	機能
0	なし	(ラベルのデフォルト)
1	細線	(テキスト ボックス，ピクチャー ボックスのデフォルト)
2	太線	(テキスト ボックスでは利用不可)

参照

Caption プロパティ，ControlBox プロパティ，MinButton プロパティ，MaxButton プロパティ

BSAVE ステートメント

BLOAD ステートメントを参照してください。

CALL ステートメント (Basic プロシージャ)

機能　Basic の SUB プロシージャに制御を渡すステートメントです。

構文　**CALL** *name*[([**BYVAL**]*argument*[, [**BYVAL**]*argument*...])]

name[[**BYVAL**]*argument*[, [**BYVAL**]*argument*...]]

name　呼び出す SUB プロシージャの名前。名前は、40 文字以内でなければなりません。

BYVAL　*argument* を参照による引き渡しではなく、値による引き渡しで渡すことを示します。配列変数に対して指定することはできません。

argument　SUB プロシージャに引き渡す、変数，配列変数，式。配列変数を渡す場合は、次のように配列名の後ろにカッコ ()を付けて指定します。

```
CALL ShellSort(IntArray())
```

解説　SUB プロシージャを呼び出すとき、**DECLARE** ステートメントでプロシージャを宣言してあれば、**CALL** キーワードを省略することもできます。**CALL** キーワードを省略するときは、引数全体を囲むカッコも省略する必要があります。**DECLARE** ステートメントで宣言せずに **CALL** キーワードを省略すると、プログラムを保管するときに、自動的に **DECLARE** ステートメントが生成されます。

1 行に複数の Basic ステートメントを記述する場合、プロシージャ名のすぐ後にステートメントの区切り文字であるコロン (:) は記述できません。このようにすると、Basic はプロシージャ名を、プロシージャへの呼び出しではなくラベルとして扱ってしまいます。

CALL ステートメントは、**DECLARE** または **SUB** ステートメントにおける引数の定義に合わせて、参照による呼び出しか、値による呼び出しのどちらかを行います。

CALL ステートメントの中では、**BYVAL** キーワードは単なるプログラムの読みやすさの意味だけで、省略してもかまいません。引数を値で渡すためには、**SUB** ステートメントまたは **DECLARE** ステートメントで、**BYVAL** キーワードを指定します。

```
DECLARE SUB Mult(BYVAL X!, Y!)
.
.
.
CALL Mult (BYVAL X!, Y!)
```

DECLARE ステートメントで **BYVAL** キーワードが指定されていない引数 (上の例の Y!) に、**CALL** ステートメント中で **BYVAL** を指定するとエラーになります。

参照　Absolute ルーチン，**CALL[S]** ステートメント (Non-Basic プロシージャ)，**DECLARE** ステートメント (Basic プロシージャ)，**SUB** ステートメント

CALL[S] ステートメント (Non-Basic プロシージャ)

機能　他の言語で書かれたプロシージャに制御を渡すステートメントです。

構文

引数が配列でないとき

CALL *name*[([{**BYVAL**|**SEG**}]*argument*[, [{**BYVAL**|**SEG**}]*argument*...])]

name[([{**BYVAL**|**SEG**}]*argument*[, [{**BYVAL**|**SEG**}]*argument*...])]

CALLS *name*[*argument*[, *argument*]...]

引数が配列のとき

CALL *name*[([*argument*[()]][, *argument*[()]...])]

name[*argument*()[, *argument*()...]]

name　呼び出す対象である、Visual Basic 以外で作成されたプロシージャの名前。名前は、40 文字以内でなければなりません。

BYVAL　*argument* を参照による引き渡しではなく、値による引き渡しで渡すことを示します。配列変数に対して指定することはできません。

SEG　*argument* をセグメント アドレス (far アドレス) として引き渡すことを示します。配列変数に対して指定することはできません。

argument　プロシージャに引き渡す Basic の変数，配列変数，式。配列変数を渡す場合は、次のように配列名の後ろにカッコ ()を付けて指定します。

```
CALL ShellSort(IntArray())
```

解説　プロシージャを呼び出すとき、**DECLARE** ステートメントでプロシージャを宣言してあれば、**CALL** キーワードを省略することもできます。**CALL** キーワードを省略するときは、引数全体を囲むカッコも省略する必要があります。**DECLARE** ステートメントで宣言せずに **CALL** キーワードを省略すると、プログラムを保管するときに、自動的に **DECLARE** ステートメントが生成されます。

CALLS ステートメントは、**CALL** のすべての引数の前に **SEG** キーワードを付けたものと同じ機能です。**CALLS** ステートメントで指定した各引数は、セグメント アドレスとして渡されます。配列変数を引数として渡したいときは、**CALLS** ステートメントは使えません。

アセンブリ言語で記述されたプロシージャに対して参照を行う場合、その名前は PUBLIC (シンボル) として宣言されている必要があります。

BYVAL キーワードを使った結果は、Basic の値による引き渡しとは異なります。

```
CALL Difference (BYVAL A,(B))
```

最初の引数である A は、その値だけが Difference に引き渡されます。一方、引数 (B) は評価されて、その値がテンポラリな記憶領域に保存されます。そして、その記憶領域のアドレスが、Difference に引き渡されます。引数を、Basic の値による引き渡しの方法でプロシージャに渡すことはできますが、テンポラリな記憶領域のアドレスが渡されるため、Basic 以外の言語のプロシージャでは、アドレスを受け取ることになることに注意してください。

Basic は、呼び出したルーチンの実行を始める前に変数のメモリ位置を変えてしまうことがあるので、配列の要素に対して **SEG** キーワードを使うときは注意してください。メモリ移動を引き起こす引数を指定すると、エラーの原因になります。**SEG** キーワードを使って変数を確実に引き渡すことができるのは、単純変数，数式，または添字として組み込み関数やユーザー定義関数を使わずに配列の要素を指定するときだけです。

参照

CALL ステートメント (Basic プロシージャ)，**DECLARE** ステートメント (Non-Basic プロシージャ)

Cancel プロパティ

機能

コマンド ボタンを、そのフォームのデフォルトの取消ボタンにするかどうかを決めます。

構文

[*formname*.]*ctlname*.**Cancel**[= *boolean%*]

ctlname	コマンド ボタンの名前。
boolean%	デフォルトの取り消しボタンであるかどうかを示す値。 True (-1) がデフォルトの取り消しボタンを示し、 False (0) がそうでないことを示します。デフォルトは False (0) です。

対象

コマンド ボタン

解説

デフォルトの取り消しボタンは、どのコントロールがフォーカスを持っていても、[ESC] キーが押されたときに Click イベントが発生するコマンド ボタンです。通常、デフォルトの取り消しボタンは、それまでユーザーが行った変更を無効にして、フォームを以前の状態に戻したりクローズしたりするために使われます。

デフォルトの取り消しボタンに対して [ESC] キーで Click イベントが発生したときは、そのときフォーカスを持っていたコントロールに対する KeyDown, KeyPress, KeyUp, Click などのイベントは発生しません。

1 つのフォーム上で、デフォルトの取り消しボタンに設定できる、つまり Cancel プロパティを True (-1) に設定できるコマンド ボタンは 1 つだけです。あるコマンド ボタンの Cancel プロパティを True (-1) に設定すると、他のすべてのボタンは自動的に False (0) に設定されます。

[ESC] キーではなく、[ENTER] キーが押されたときに Click イベントが発生するコマンド ボタンを設定するには、Default プロパティを使います。

参照

Click イベント，Default プロパティ，KeyDown イベント，KeyPress イベント，KeyUp イベント

Caption プロパティ

機能

コントロール上やフォームのタイトル バーに表示されるテキストを決めます。

構文

[*formname.*][*ctlname.*]**Caption**[= *string-expression$*]

string-expression$ 表示されるテキストを示す文字列式。デフォルトは、 FormName プロパティまたは CtlName プロパティの値と同じです。

対象

チェック ボックス，コマンド ボタン，フォーム，フレーム，ラベル，メニュー，オプション ボタン

解説

Caption プロパティに設定されているテキスト (キャプション) は、フォームの場合はタイトル バー、コントロールの場合はコントロールの上に表示されます。

デフォルトで Caption プロパティに設定される値は、フォーム名 (FormName プロパティの値) またはコントロール名 (CtlName プロパティの値) と同じです (Form1, Label1, Command1 など)。

フォームのタイトル バーが表示されるかどうかは、フォームの BorderStyle プロパティの設定によります。BorderStyle プロパティが 0 に設定されている場合は、タイトル バーは表示されません。

Caption プロパティに長いテキストを設定しても、そのときフォームのタイトル バーやコントロールで表示できる長さを超える部分は表示されません。

コントロールに対するアクセス キーを割り当てるには、Caption プロパティに文字列を設定するときに、アクセス キーに設定したい文字の前に "&" を付けます。アクセス キーに設定できるのは 1 つの項目に対して 1 文字だけです。また、2 バイト文字をアクセス キーにすることはできません。アクセス キーの表示色は、**SCREEN** オブジェクトの ControlPanel プロパティの ACCESSKEY_FORECOLOR 要素によって決まります。

ラベル コントロールのサイズをキャプションの大きさに合わせて自動的に変更したい場合は、AutoSize プロパティを True (-1) に設定します。

参照　Text プロパティ

CDBL$ 関数

機能　文字列中の 1 バイト系文字を、2 バイト系文字に変換して返します。

構文　**CDBL$**(*string-expression$*)

string-expression$　変換元の文字列。

解説　*string-expression$* には、どんな文字列式を指定してもかまいません。**CDBL$** 関数は、指定された *string-expression$* 中の 1 バイト系文字を 2 バイト系文字に変換した結果を返します。文字列中に含まれた 2 バイト系文字と、変換対象外の 1 バイト系文字は変換されません。

この関数で変換される 1 バイト系文字は、すべての英数字、カタカナ，スペースと、以下の記号です。

! " # $ % & ' ( ) * + , - . / : ; < = >

? @ [¥] ^ _ ` { | } ~ 。「 」、・ー ゛ ゜

これらは、ASCII コード &H20～&H7E および &HA1～&HDF の文字すべてです。

参照　**CSNG$** 関数，**CDBL** 関数

CHAIN ステートメント

機能

他のプログラムに制御を渡すプログラム制御ステートメントです。

構文

CHAIN *filespec$*

filespec$　制御を渡すプログラムを示す文字列。パス名も指定できます。

解説

プログラミング環境で **CHAIN** ステートメントを実行する場合は、拡張子 .BAS のファイルに対してのみチェインできます。この場合、拡張子は省略できます。拡張子 .EXE や .COM のファイルにはチェインできません。

プログラミング環境外で **CHAIN** ステートメントを実行する場合は、拡張子 .EXE のファイルに対してのみチェインでき、拡張子 .BAS のファイルにチェインすることはできません。

COMMON ステートメントを使って空の COMMON ブロックを設定すると、複数のプログラム間で変数を引き渡すことができます。

/O オプションを指定してコンパイルした独立型の実行可能ファイルでは、**COMMON** を利用できません (VBDCL10.LIB ライブラリは **COMMON** をサポートしていません)。プログラミング環境外でチェインされるプログラム間で **COMMON** を使うには、次のどちらかの方法で行う必要があります。

- コンパイル時に、[実行ファイル作成] ダイアログ ボックスから、"ランタイム分離型実行可能ファイル" オプションを選択します。この設定がデフォルトです (VBDRT10.EXE を使います)。
- コマンドラインから、/O オプションを付けずに BC.EXE を実行してコンパイルします。

ランタイム ライブラリ (VBDRT10.EXE) を使う分離型の実行可能ファイルで **CHAIN** ステートメントを使うときは、ファイルは、**CLOSE** ステートメントを使って明確にクローズしない限り、チェインが行われている間オープンされたままになります。

CHAIN ステートメントと **RUN** ステートメントは似た機能を持っています。主な違いは、**RUN** ステートメントがオープンされているすべてのファイルをクローズすることと、COMMON データブロックをサポートしていないことです。

/O オプションを指定してコンパイルした独立型の実行可能ファイルでは、**CHAIN** は **RUN** と同じ機能です。

/O オプションを指定せずにコンパイルしたフォームを使わないプログラムから、/O オプションを指定せずにコンパイルしたフォームを使うプログラムにチェインしようとすると、エラーが発生します.

参照

CALL ステートメント (Basic プロシージャ), **COMMON** ステートメント, **RUN** ステートメント, **SHELL** ステートメント

Change イベント

機能

コントロールの内容が変化したときに発生します。

構文

SUB *ctlname*_**Change** ([*Index* **AS INTEGER**])

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，水平スクロール バー，ラベル，テキスト ボックス，垂直スクロール バー

解説

Change イベントは、コントロール間で同期を取ったり、表示されたデータの間の調整を行うのに使われます。例えば、スクロール バーの Change イベントに対するイベント プロシージャを使って、スクロール バーの Value プロパティの値を表示するラベルを更新する処理を記述できます。また、作業領域にデータと数式を表示し、別の領域に計算結果を表示する処理などにも使います。

Change イベントは、ファイル システムに関連するコントロールの各プロパティを更新するときにも便利です。例えば、Change イベントを使って、ドライブ リスト ボックスの Drive プロパティが変更されたら、ディレクトリ リスト ボックスの Path プロパティにそれを反映させることができます。

Change イベントが発生する条件とタイミングは、コントロールによって異なります。

コントロール	説明
コンボ ボックス	ユーザーがデータを入力するか、ﾄﾞﾛｯﾌﾟﾀﾞｳﾝ ｺﾝﾎﾞ (Style = 0) または ｼﾝﾌﾟﾙ ｺﾝﾎﾞ (Style = 1) の内容が変更された。
テキスト ボックス	ユーザーがデータを入力するか、コード内で Text プロパティが変更された。
ディレクトリ リスト ボックス	ユーザーが新しいディレクトリを選択するか、コード内で Path プロパティが変更された。
ドライブ リスト ボックス	ユーザーが新しいドライブを選択するか、コード内で Drive プロパティが変更された。
スクロール バー	ユーザーがスクロールを行うか、コード内 で Value プロパティが変更された。
ラベル	コード内で Caption プロパティが変更された。

Change イベントによって、イベントが連鎖反応を起こし、処理がループしてしまう場合があります。これは、あるコントロールの Change イベントに対するイベント プロシージャによって、結果的にそのコントロール自身の内容が変更されるしまうような場合に起こります。例えば、テキスト ボックスの Text プロパティの設定のように、コントロールの値を左右するプロパティをコードに記述する場合がこれに相当します。イベントの連鎖反応を防ぐには、次のようにしてください。

- 可能であれば、そのコントロールの内容を変更するような Change イベント プロシージャを記述しないようにしてください。さもなければ、1 つの Change イベント プロシージャの実行中に、もう 1 度そのイベント プロシージャが実行されないように、フラグ変数を設定してください。
- それぞれの Change イベント プロシージャが相互に影響し合うような複数のコントロールを作成するのは避けてください。例えば、互いに相手を更新し合う 2 つのテキスト ボックスなどがこれに相当します。

参照

KeyDown イベント，KeyPress イベント，KeyUp イベント，PathChange イベント，PatternChange イベント

CHDIR, MKDIR, RMDIR, FILES ステートメント

機能

CHDIR ステートメントは、指定したドライブのデフォルトのディレクトリを変更します。**MKDIR** ステートメントは、新しいディレクトリを作成します。**RMDIR** ステートメントは、指定したディレクトリを削除します。**FILES** ステートメントは、指定したドライブやディレクトリ上にあるファイルの一覧を表示します。**FILES** ステートメントは、フォームが表示されている状態では利用できません。

構文

CHDIR *pathname$*

MKDIR *pathname$*

RMDIR *pathname$*

FILES [*filespec$*]

pathname$	新しくデフォルトに設定するディレクトリや，作成または削除するディレクトリのパス名を示す文字列式。
filespec$	一覧を表示するドライブやディレクトリ、または特定のファイルを示す文字列式。疑問符 (?) やアスタリスク (*) などの MS-DOS のワイルドカード文字も使えます。省略すると、カレント ディレクトリと見なされます。

解説

CHDIR, MKDIR, RMDIR ステートメント

CHDIR, MKDIR, RMDIR ステートメントは、それぞれ MS-DOS の同名のコマンドと同じ機能です。ただし省略して CD, MD, RD とすることはできません。

対象のディレクトリは、次の構文にしたがい、64 文字以内の文字列で指定します。

[*drive*:][¥]*directory*[¥*directory*]

絶対パス，カレント ディレクトリからの相対パスのどちらで指定してもかまいません。"." はカレント ディレクトリ，".." はその親ディレクトリを示します。

CHDIR ステートメントは、指定されたドライブのデフォルトのディレクトリを変更します。ただしカレント ドライブは変更しません。例えば、カレント ディレクトリが C: ドライブにあるとき、次の **CHDIR** ステートメントは、D: ドライブのデフォルトのディレクトリを D:¥TMP に変更しますが、カレント ドライブは C: のままです。

```
CHDIR "D:¥TMP"
```

あるドライブのデフォルトのディレクトリを知るには、**CURDIR$** 関数を使います。また、カレント ドライブを変更するには **CHDRIVE** ステートメントを使ってください。

MKDIR ステートメントはディレクトリを作成しますが、ディレクトリ名の中にスペースが入っているかどうかのチェックはしていませんので注意してください。もしスペースを含む名前のディレクトリを作成した場合、MS-DOS を通してそのディレクトリにアクセスすることはできますが、削除するには Visual Basic の **RMDIR** ステートメントを使う必要があります。

RMDIR ステートメントは指定したディレクトリを削除しますが、中にそのディレクトリ自身 (.) と親ディレクトリ (..) 以外のファイルがあると、"パスが見つかりません" または "パス名が無効です" エラーになります。

FILES ステートメント

FILES ステートメントは指定したディレクトリ中のファイルの一覧を表示しますが、疑問符 (?) やアスタリスク (*) などの OS のワイルドカード文字も使うことができます。

FILES ステートメントは、フォームが表示されている状態では利用できません。フォームを使うアプリケーションであるディレクトリ中のファイルの一覧を表示するには、ディレクトリ リスト ボックスを使ってください。

CHDRIVE ステートメント

機能

カレント ドライブを変更します。

構文

CHDRIVE *drive$*

drive$ 新しく設定するカレント ドライブを示す文字列式。2 文字以上の文字列を指定した場合は、最初の 1 文字だけが使われます。

解説

drive$ で指定するドライブは、必ず存在していなければなりません。存在しない場合はエラーになります。指定できる範囲は、A から *lastdrive* までです。*lastdrive* は、CONFIG.SYS ファイルで設定可能な、最後のドライブ名です。

drive$ に null 文字列 ("") を指定すると、カレント ドライブは変更されません。

大文字小文字は区別されません ("C" は "c" と同じです)。

常に *drive$* に指定した最初の文字だけが使われるので、コロン (:) を付ける必要はありません。

Checked プロパティ

機能

メニュー コマンドの横にチェック マークが表示されるかどうかを決めます。

構文

[*formname*.]*ctlname*.**Checked**[= *boolean%*]

ctlname	メニュー コントロールの名前。
boolean%	チェック マークが表示されるかどうかを示す値。 True (-1) は表示されることを示し (チェック状態)、False (0) は表示されないことを示します (未チェック状態)。デフォルトは False (0) です。

対象

メニュー

解説

チェック マークは、メニュー項目の左に表示されるマークで、オン/オフを切り換えるメニュー コマンドなどで、その項目がオンになっていることをユーザーに示すために使います。通常は Click イベントに対するイベント プロシージャでチェック/未チェックを切り換えます。

トップ レベルのメニュー項目には、チェック マークを付けることはできません。

参照

Enabled プロパティ，Separator プロパティ，Visible プロパティ

CHR$ 関数

ASC 関数を参照してください

CIRCLE ステートメント

機能

指定した中心と半径で、楕円または円を描きます。フォームが表示されている状態では利用できません。

構文

CIRCLE [**STEP**](*x!*,*y!*), *radius!*[, [*color%*][, [*start!*][, [*end!*][, *aspect!*]]]]

(*x!*,*y!*)	円または楕円の中心のビューポート座標。
STEP	座標を相対座標で指定することを示します。
radius!	半径を示す数式。楕円を描く場合は、 x または y の長い方の半径を示します。
color%	楕円や円の色を示すパレット番号。省略すると、グラフィックスのデフォルトの色と見なされます。
start!	ラジアン単位の弧の開始角。

end!　ラジアン単位の弧の終了角。

aspect!　縦横比、つまり y 方向の半径と x 方向の半径の比率を示す値。

解説

フォームが表示されているときに **CIRCLE** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **CIRCLE** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

ビューポート座標は、**WINDOW** ステートメントが実行されていなければ物理座標と同じです。物理座標は、**VIEW** ステートメントが実行されていなければ、画面左上角を原点とする絶対座標です。

STEP キーワードは、現在のグラフィックス出力位置 (直前にプロットされた点) からの相対座標であることを示します。例えば、グラフィックス出力位置が (10,10) のときに STEP (4, 5) と指定すると、座標 (10+4, 10+5) すなわち (14,15) を意味します。

構文の途中の引数を省略することはできますが、省略したことを示すカンマ (,) を必ず入力してください。逆に最後をカンマで終わるとエラーになります。次のステートメントは、*color%* 引数が省略されています。

```
CIRCLE STEP(150,200),94,,0.0,6.28
```

円や楕円を描き終ると、グラフィックス出力位置は円または楕円の中心に移動します。

中心の座標は、画面やビューポートの外にある点でもかまいません。

start! と *end!* 引数には、弧を描くときの開始点と終了点の角度を、右端を 0 としてラジアン単位で指定します。弧は、反時計回りに描かれます。指定できる範囲は -2π～2π で、*start!* のデフォルトの値は 0, *end!* のデフォルトの値は 2π です (π≒3.141593)。*start!* に *end!* より小さい角度を指定することもできます。*start!* か *end!* に負の数を指定すると、**CIRCLE** ステートメントは、その点と中心を結ぶ直線 (半径) を描きます (両方負の値にすると扇型)。このとき実際の角度としては、指定した数の絶対値が使われます。ただし、*start!* または *end!* に 0 を指定して半径を描くときは、次に示すように非常に小さな負の値を指定してください。-0 と指定しても半径は描かれません。

```
' 円の 1/4 の扇型を描く
SCREEN 2
CIRCLE(200,100),60,,-.0001,-1.57
```

aspect! 引数には縦横の比率を指定します。

radius! は、*aspect!* が 1 より小さいときは x 軸半径、*aspect!* が 1 より大きいときは y 軸半径を示します。

参照

SCREEN ステートメント，**LINE** ステートメント

CLEAR メソッド

機能

CLIPBOARD オブジェクトの内容をクリアします。

構文

CLIPBOARD.CLEAR

対象

CLIPBOARD オブジェクト

解説

CLIPBOARD オブジェクトとそれに適用可能なメソッドを使うと、コード中から **CLIPBOARD** オブジェクトの内容を操作できます。テキスト ボックスやコンボ ボックスの SelStart, SelLength, SelText プロパティと組み合わせて使うことによって、メニューやコマンド ボタンでカット & ペーストの機能を実現できます。

キーワード **CLIPBOARD** は省略できません。フォーム モジュール中で省略するとエラーが発生します。また、コード モジュール中で単独で **CLEAR** キーワードを使うと、**CLEAR** ステートメントと見なされます。

参照

CLEAR ステートメント

CLEAR ステートメント

機能

引数が指定されていないときの **CLEAR** ステートメントは、次の処理を行います。

- すべてのファイルをクローズします。
- すべてのファイル バッファを解放します。
- すべての COMMON 変数を消去します。
- すべての数値変数と配列を 0 に設定します。
- スタックを初期化します。
- すべての文字列変数を null 文字列に設定します。
- *stacksize&* が指定されると、**CLEAR** ステートメントは、スタック サイズの変更も行います。

構文

CLEAR [,,*stacksize&*]

stacksize&	バイト単位のスタック サイズ。前に 2 つのカンマ (,) を指定することに注意してください。

解説

再帰呼び出しを行うプロシージャなどによってプログラム中のサブルーチンやプロシージャのネストが深くなると、**CLEAR** ステートメントを使用してスタックのサイズを増やす必要が生ずることがあります。また、プロシージャの引数が多い場合も、スタック サイズを増やす必要があります。

GOSUB ステートメントの実行中にスタックをクリアすると、スタックに置かれたリターンアドレスが消去されてしまいます。消去すると、**RETURN** ステートメントを正しく実行することができなくなり、ランタイム エラー "GOSUB がありません" が発生します。また、SUB または FUNCTION プロシージャの中で **CLEAR** ステートメントを使うと、ランタイム エラー "引数が有効範囲にありません" が発生します。

参照

CLEAR メソッド，**STACK** 関数，**STACK** ステートメント

Click イベント

機能

フォームまたはコントロールがクリックされたときや、同じ意味のキーボード操作が行われたとき、または同じ意味の操作がコード中から行われたときに発生します。

構文

SUB Form_Click ()

SUB *ctlname*_**Click** ([*Index* **AS INTEGER**])

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ファイル リスト ボックス，フォーム，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス

解説

フォームの場合は、フォーム上のコントロールが配置されていない領域を、ユーザーがマウスの左ボタンでクリックしたときに、Click イベントが発生します。

コントロールの場合は、次に示す操作が行われたときに、Click イベントが発生します。

- コントロールがマウスの左ボタンでクリックされた。
- 方向キーやマウスの左ボタンで、コンボ ボックスまたはリスト ボックスの項目が選ばれた。
- コマンド ボタンまたはチェック ボックスがフォーカスを持つ状態で、スペースキーが押された。
- Default プロパティが True (-1) に設定されているコマンド ボタンが存在するフォームで、[ENTER] キーが押された。
- Cancel プロパティが Ture (-1) に設定されているコマンド ボタンが存在するフォームで、[ESC] キーが押された。
- コントロールに割り当てられているアクセス キーが押された。
- 方向キーやマウスで、グループ化されているオプション ボタンの別のボタンが選択された。

- コード中から、ListIndex プロパティが新たに負でない値に設定された。
- コード中から、コマンド ボタンの Value プロパティが True (-1) に設定された。
- コード中から、オプション ボタンの Value プロパティが True (-1) に設定された。
- コード中から、チェック ボックスの Value プロパティが変更された。

通常、コマンド ボタン，メニュー，ピクチャー ボックスに対しては、Click イベントに対応するイベント プロシージャを記述します。

マウスの左右のボタンを区別するときは、MouseDown と MouseUp イベントを使います。

コントロールの Value プロパティを使うと、コード内のコントロールの状態を調べることができます。

フォーム，ディレクトリ リスト ボックス，ファイル リスト ボックス，リスト ボックス，ラベル，ピクチャー ボックスに対して Click イベントが発生するときは、同時に MouseDown, MouseUp イベントも発生します。発生する順番は、MouseDown, MouseUp, Click です。

メニュー タイトルまたはトップ レベルのメニュー項目がクリックされると、その次のレベルのメニューが自動的にドロップ ダウンします。このとき、Click イベントも同時に発生します。このイベントに対するイベント プロシージャを記述しておけば、表示するメニューの内容を変更できます。ただし、このイベント プロシージャで、Paint などの他のイベントを発生させないようにしてください。

参照　DblClick イベント，Default プロパティ

CLIPBOARD オブジェクト

機能　**CLIPBOARD** オブジェクトは特殊オブジェクトの 1 つで、**CLIPBOARD** キーワードを指定することによってアクセスされます。テキスト ボックスやコンボ ボックスでカット & ペーストを行うときに、コード中からクリップボードの内容を操作するために使います。

構文　**CLIPBOARD.***method*

method　**CLIPBOARD** オブジェクトに適用可能なメソッド

解説

メソッド

CLEAR　GETTEXT　SETTEXT

CLIPBOARD オブジェクトは、クリップボードを示す特殊オブジェクトです。テキスト ボックスとコンボ ボックスのテキスト ボックス部分には、あらかじめ Microsoft Windows と同様のカット & ペーストの機能が用意されています。つまり、

[CTRL]+[INS] または [CTRL]+[C] でクリップボードへのコピー (複写), [SHIFT]+[DEL] または [CTRL]+[X] でクリップボードへのカット (切り抜き), [SHIFT]+[INS] または [CTRL]+[V] でクリップボードからのペースト (貼り付け) が行えます。この機能は、1 つのテキスト ボックスまたはコンボ ボックス内だけでなく、複数のテキスト ボックスまたはコンボ ボックス間でも有効です。**CLIPBOARD** オブジェクトは、クリップボードの内容をコード中から操作したいときに使います。

クリップボード (つまり **CLIPBOARD** オブジェクト) に保持できるテキストは、1 度に 1 セットだけです。他のテキストが置かれると、古いテキストは失われます。

CLIPBOARD オブジェクトは、テキスト ボックスやコンボ ボックス以外のコントロールに対しても利用できます。この場合、他のコントロール用にメソッドを組み合わせてプログラミングしてください。

CLOSE ステートメント

機能

ファイルまたはデバイスへの入出力を終了します。

構文

CLOSE[[#]*filenumber%*[, [#]*filenumber%*]...]

filenumber% クローズするファイルまたはデバイスのファイル番号。

解説

指定するファイル番号は、そのときオープンされているファイルまたはデバイスに割り当てられていなければなりません。クローズできるとファイル番号は解放され、同じまたは別のファイル, デバイスのオープンに使うことができます。また、使用されていたすべてのバッファ領域が解放されます。

引数をすべて省略すると、オープンされているすべてのファイルおよびデバイスがクローズされます。

シーケンシャル出力モードでオープンされているファイルやデバイスに対して **CLOSE** ステートメントを実行すると、出力バッファに残っているデータが、そのファイルやデバイスに書き込まれます。

CLEAR, END, RESET, RUN, SYSTEM の各ステートメントも、すべてのファイルおよびデバイスをクローズします。また、**/O** オプションを指定してコンパイルした独立型の実行可能ファイルでは、**CHAIN** ステートメントを実行しても、すべてのファイルとデバイスがクローズされます。

CLS メソッド

機能

フォームまたはピクチャー ボックスに対して実行時に出力されたテキストをクリアします。

構文

[{{*formname.*]*ctlname.*|*formname.*}}**CLS**

ctlname	ピクチャー ボックスの名前。

対象

フォーム，ピクチャー ボックス

解説

CLS メソッドは、実行時に **PRINT** メソッドで出力されたテキストをクリアします。フォーム上に置かれたコントロールや背景のイメージはクリアされません。

フォーム モジュールで、(対象のオブジェクトを指定せずに) **CLS** キーワードを単独で使うと、フォームに対する **CLS** メソッドと解釈されます。これに対してコード モジュール中では、**CLS** ステートメントと解釈されます。

フォームまたはピクチャー ボックスの CurrentX および CurrentY プロパティ (次の出力位置) は、**CLS** メソッドの後では 0 にリセットされます。

フォームやピクチャー ボックスに、AutoRedraw プロパティが True (-1) に設定されている間に表示されたテキストは、**CLS** メソッドを使う前に AutoRedraw プロパティを False (0) に変更すると、クリアされません。ただしこの場合でも、次の出力位置はフォームやピクチャー ボックスの左上に再設定されますから注意してください。また、AutoRedraw プロパティを再び True (-1) に戻してから **CLS** メソッドを使えばクリアされます。AutoRedraw プロパティを細かく操作すれば、クリアされるものとされないものを出力時に区別できます。

参照

CLS ステートメント

CLS ステートメント

機能

ディスプレイ画面をクリアします。フォームが表示されている状態では利用できません。また、フォーム モジュール中では使用できません。**CLS** メソッドを使ってください。

構文

CLS [0|1|2]

0	表示されているすべてのテキストとグラフィックスをクリアすることを示します。
1	グラフィックス ビューポート内をクリアすることを示します。
2	テキスト ビューポート内をクリアすることを示します。

省略　グラフィックス ビューポート内とテキスト ビューポート内をクリアすることを示します。

解説

CLS がステートメントとして解釈されるか **CLS** メソッドと解釈されるかは、キーワード **CLS** がコード中のどの位置で使われているかによって決まります。フォーム モジュールでは、**CLS** キーワードは **CLS** ステートメントではなく **CLS** メソッドと解釈されます。**CLS** キーワードがフォーム モジュール中のジェネラル プロシージャ中にあっても同じです。つまり、**CLS** がステートメントとして解釈されるのは、コード モジュール中で、オブジェクトを指定せずに単独で **CLS** と記述されている場合だけです。

フォーム モジュールから **CLS** ステートメント特有の機能を利用するには、**CLS** ステートメントをコード モジュール中のジェネラル プロシージャに入れて、それを呼び出すようにします。

フォームが表示されているときに **CLS** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

カーソルは、テキストビューポート左上隅に移動します。グラフィックスの出力位置は画面中央に設定されます。

参照

CLS メソッド，**VIEW PRINT** ステートメント，**VIEW** ステートメント，**WINDOW** ステートメント

COLOR ステートメント

機能

文字やグラフィックスの色と背景色を設定します。フォームが表示されている状態では利用できません。

構文

COLOR [*foreground*][, [*background*]]

foreground　文字の色やグラフィックスのデフォルトの色を示す値。デフォルトは 15 (白 (明)) です。

background　文字の背景色を示す値。デフォルトは 0 (黒) です。

解説

フォームが表示されているときに **COLOR** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

フォームを使用しているときは、画面の各部の色は ControlPanel, BackColor, ForeColor の各プロパティを使って設定します。

COLOR ステートメントを使うと、フォームを使わない場合の文字や文字の背景色、およびグラフィックス ステートメントで図形を描くときのデフォルトの色を設定できます。

COLOR ステートメントの機能，指定できる色，指定方法は、ハードウェアやそのときのスクリーン モードに依存します。『機種依存情報』を参照してください。

参照　**SCREEN** ステートメント，**PALETTE** ステートメント

COM ステートメント

機能

指定したポートでの通信イベントのトラップを有効，無効，中断状態にします。フォームやコントロールに関連したイベントには影響を与えません。

構文

COM(*port%*) {**ON**|**OFF**|**STOP**}

port%	シリアル ポートの番号 (1 または 2)。
ON	*port%* で指定したポートのイベント トラップを有効にすることを示します。**ON COM** ステートメントによる通信イベントのトラップが可能になります。
OFF	*port%* で指定したポートのイベント トラップを無効にすることを示します。このステートメントが実行されたあとに起こったイベントは記憶されません。
STOP	*port%* で指定したポートのイベント トラップを中断することを示します。このステートメントが実行されたあとでイベントが発生すると、そのイベントは記憶されます。次に **COM**(*port%*) **ON** ステートメントが実行されると、すぐに **ON COM** ステートメントでトラップされます。

解説

このステートメントで扱うイベントは、フォームやコントロールを使ったイベント ドリブン プログラミング モデルのイベントとは異なることに注意してください。フォームやコントロールに関連したイベントには影響を与えません。ただし、フォームやコントロールを使ったアプリケーションでも使うことができます。

イベント トラップが有効であるとき、そのイベントが発生していないかどうか (通信ポートにデータが到着していないかどうか) が自動的に監視されています。このチェックは、プログラミング環境で実行されている場合と、/V オプション付きでコンパイルされた実行可能ファイルでは、ステートメントごとに行われます。/W オプション付きでコンパイルされた実行可能ファイルでは、行番号または行ラベルごとにチェックされます。イベントが発生していると、**ON COM** ステートメントで設定されているイベント処理ルーチンに制御が移ります。

イベント トラップを行うステートメントを含むプログラムを、/W または /V オプションを指定せずにコンパイルすると、"ON event ステートメントを含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

COM(*port%*) **STOP** ステートメントでトラップを中断している間に起きたイベントは、**COM**(*port%*) **OFF** ステートメントが実行されると失われます。

イベントがトラップされ、対応する処理ルーチンに制御が移ると、再帰的なトラップが起こらないように自動的に **COM**(*port%*) **STOP** が実行されます。処理ルーチンが終了すると、そのルーチンの中で明確に **COM**(*port%*) **OFF** が実行されていない限り、自動的に **COM**(*port%*) **ON** ステートメントが実行されます。

参照

ON COM ステートメント，**OPEN COM** ステートメント，**PRINTER** オブジェクト，**UEVENT** ステートメント

COMMAND$ 関数

機能

Visual Basic で開発されたアプリケーションの起動のときに指定されたコマンド ラインの引数部分を含む文字列を返します。

構文

COMMAND$

解説

Visual Basic で作成した実行可能ファイルでは、**COMMAND$** 関数は、起動時にアプリケーション名の後に指定されたすべての引数を返します。例えば、次の例では cmdlinearg が返されます。

```
MYAPP cmdlinearg
```

プログラミング環境上で実行するときには、[実行] メニューの [COMMAND$ 編集] コマンドを使うか、VBDOS.EXE を起動するときに /CMD オプションを指定すると、**COMMAND$** 関数の戻り値を指定できます。例えば、次のように Visual Basic を起動すると、**COMMAND$** 関数は、cmdlinearg を返します。

```
VBDOS /CMD cmdlinearg
```

COMMAND$ は、コマンド ラインのすべての文字を大文字に変換し、先頭の空白文字を削除します。

COMMON ステートメント

機能

モジュール間や、チェインしたプログラムの間で共有する、グローバル変数を宣言します。

構文

COMMON[**SHARED**][*/blockname/*]*variable*[()][**AS** *type*][, *variable*[()][**AS** *type*]]...

SHARED
宣言された変数をモジュール内のすべてのプロシージャで共有することを示します。変数は、そのモジュールに対してグローバルになります。プロシージャの中で、個々に **SHARED** ステートメントを使う必要はありません。**SHARED** を指定できるのは、モジュールレベル コード内だけです。

/blockname/
変数のグループを示す識別子。いくつかの変数をグループ化し、それらを共有するときに指定します。*blockname* を指定すると、COMMON ブロックに名前が付けられます。省略すると、名前のない COMMON ブロックになります。型宣言文字 (%, &, !, #, @, $) を含めることはできません。

variable
宣言する変数の名前。配列を指定するときは、空のカッコを後ろに付けます。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。

type
変数のデータ型 (**INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, **STRING** * *n* またはユーザー定義型)。変数それぞれに対して指定します。省略すると、変数名に型宣言文字が付けられていればそのデータ型になり、付けられていなければ **SINGLE** (単精度浮動小数点数) と見なされます。

解説

COMMON ステートメントは、変数を記憶するための特別なメモリ領域 (COMMON ブロック) を確保し、複数のモジュール間や、**CHAIN** ステートメントでチェインするプログラム間で変数を共有するために使います。1 つのモジュール内で変数を共有するには、**COMMON** ステートメントではなく、**SHARED** キーワードを付けた **DIM** ステートメントを使います。

宣言の方法

例えば、複数のモジュール間で、整数型の単純変数 MyValiable を共有したいとき、各モジュールのモジュール レベルの先頭で次のように宣言します。

```
COMMON SHARED MyValiable AS INTEGER
```

このステートメントは、変数を共有するすべてのモジュールのモジュール レベルに記述する必要があります。モジュール間で **COMMON** ステートメントの宣言が一致していないと、発見しにくいエラーの原因となります。**COMMON** ステートメントをインクルード ファイルに記述し、各モジュールで **$INCLUDE** メタコマンドを使って、そのファイルをインクルードするようにすると便利です。

```
' インクルード ファイル COMDEF.BI
COMMON SHARED MyValiable AS INTEGER
```

```
' MAINFORM.FRM のモジュール レベル
'$INCLUDE:'COMDEF.BI'
.
.
.

' SUBFORM1.FRM のモジュール レベル
'$INCLUDE:'COMDEF.BI'
.
.
.
```

モジュール間で配列変数を共有するときには注意が必要です。**COMMON** ステートメントでは、配列変数を指定するとき、空のカッコ () を付けて配列変数名を指定するだけです。配列の次元や添字の上限，下限は、**DIM** または **REDIM** ステートメントで別に宣言します。

配列には、動的配列と静的配列があります。

動的配列を共有したい場合は、**COMMON** ステートメントより後で、**REDIM** または **DIM** ステートメントで宣言します。このとき、**REDIM** または **DIM** ステートメントは実行可能ステートメントであるため、フォーム モジュールのモジュール レベルには置けないことに注意してください。この場合、**REDIM** または **DIM** ステートメントはインクルード ファイルに置かずに、メインのフォームの Load イベントに対するイベント プロシージャの先頭に置くようにしてください。

```
COMMON SHARED MyArray() AS INTEGER
DIM MyArray(1 TO 100)          ' コード モジュールのモジュール レベル
                               ' または Load イベントに対するイベント
                               ' プロシージャの先頭
```

動的配列の宣言

静的配列を共有したい場合は、**COMMON** ステートメントより前に、**DIM** ステートメントで添字を定数で指定して宣言します。例えば、複数のモジュール間で、添字の下限が 1，上限が 100 の整数型の静的な配列変数 MyArray を共有したいとき、各モジュールの先頭に次のように記述します。

```
DIM MyArray(1 TO 100)
COMMON SHARED MyArray() AS INTEGER
```

COMMON ステートメントで指定した変数はプログラム全体に対してグローバルな変数となるため、**COMMON** ステートメントはすべての実行可能ステートメントよりも前になければなりません。実行可能ステートメントとは、次のものを除くすべてのステートメントです。

COMMON	**CONST**	**DATA**	**DECLARE**
DEFtype	**DIM** (静的配列)	**$DYNAMIC**	**$FORM**
$INCLUDE	**OPTION BASE**	**OPTION EXPLICIT**	**REM** (メタコマンド)
SHARED	**$STATIC**	**STATIC**	**TYPE...END TYPE**

DIM ステートメントは、動的配列を宣言するときは実行可能ステートメントとして扱われます。

プログラムをチェインするとき、チェイン元のプログラムが配列を静的配列として扱っているのなら、チェイン先のプログラムでも同様に扱わなければなりません。この関係は、動的配列でも同じです。

動的配列を **COMMON** ステートメントを使って共有する場合、確保された **COMMON** ブロックに割り当てられるのは配列識別子だけで、実際の各要素の値は格納されません。

COMMON ステートメントで, 64KB を超えるサイズの配列を扱うことはできません。64KB を超えるサイズの配列を宣言することは、絶対に避けてください。

宣言の順番とデータ型

COMMON ステートメントは、各モジュールやチェインされるプログラムごとに管理されているメモリと別の領域 (COMMON ブロック) を確保し、そこに変数を格納することによって、変数を共有します。

COMMON ブロック内の領域と各モジュールやプログラム中の変数は、変数名で結び付けられるのではなく、宣言された順番とデータ型によって結び付けられます。これは、各モジュールやプログラムで変数名が異なっていても、同じ値を共有できることを意味します。また、共有する変数の数が、すべてのモジュールやプログラムで一致している必要もありません。例えば、すべてのモジュールで共有する変数は 1 つだけで、ある 2 つのモジュールだけ、もう 1 つ別の変数を共有するようにすることもできます。この場合、**COMMON** ステートメントでは、すべてのモジュールで共有される変数を先に指定しなければなりません。次の例は、**COMMON** ステートメントの中の変数が名前ではなく順序で結合されている様子を示しています。

```
COMMON SHARED A%, D$, E%          ' モジュール 1
A% = 5 : D$ = "test" : E% = 10
.
.
.
COMMON SHARED A%, E$, D%          ' モジュール 2
.                                 ' A% = 5, E$ = "test", D% = 10
.
.
COMMON SHARED A%, D$              ' モジュール 3
.                                 ' A% = 5, D$ = "test"
.
.
```

ただし、Visual Basic のプログラムとユーザーが作成したライブラリ内のルーチンで、**COMMON** ステートメントを使って変数を共有するとき、Visual Basic のプログラムで、より多くの変数を宣言することはできません。

名前付きの COMMON ブロック

/blockname/ 引数を指定すると、モジュール間で共有する変数をグループに分けることができます。適切にグループ分けすれば、それぞれのモジュールで、そのモジュールに必要な変数だけを共有することができます。*/blockname/* 引数を指定して変数を宣言すると、確保される COMMON ブロックには、*/blockname/* 引数で指定した名前が付けられます。このような COMMON ブロックを、名前付きの COMMON ブロックと呼びます。

次のプログラムでは、直方体の体積と密度を計算しています。このプログラムは、名前付きの COMMON ブロックを使い、2 つの SUB プロシージャで複数の変数を共有しています。Volume プロシージャは、直方体の辺の長さを表す変数だけを共有しています (Sides という名前の COMMON ブロック)。Density プロシージャは、さらに重さを表す変数を共有しています (Weight という名前の COMMON ブロック)。

```
' 実行開始モジュール A
DIM S(3)
COMMON /Sides/ S()
COMMON /Weight/ C
C=52
S(1)=3:S(2)=3:S(3)=6
CALL Volume
CALL Density
END
' モジュール B
DIM S(3)
COMMON SHARED /Sides/ S()
STATIC SUB Volume       'Volume という名前の SUB プロシージャ
Vol=S(1)*S(2)*S(3)
.
.
.
END SUB
' モジュール C
DIM S(3)
COMMON SHARED /Sides/ S()
COMMON SHARED /Weight/ W
STATIC SUB Density      'Density という名前の SUB プロシージャ
Vol=S(1)*S(2)*S(3)
Dens=W/Vol
.
.
.
END SUB
```

チェインを行うときの方法

名前付きの COMMON ブロックは、プログラムをチェインするときには利用できません。チェインされたプログラムに変数を引き渡すには、*/blockname/* 引数を指定せずに **COMMON** ステートメントを使ってください。

/O オプションを指定してコンパイルした独立型の実行可能ファイルでは、**COMMON** ステートメントを使ってチェイン先のプログラムに変数を渡すことはできません。チェインする場合に変数の受け渡しを行いたいときは、ランタイム モジュールを利用する分離型の実行可能ファイルを作成してください。

変数を引き渡すには、各プログラムに **COMMON** ステートメントが含まれている必要があります。この場合、変数を指定する順序とデータ型は、チェインを行うすべてのプログラムの **COMMON** ステートメントで一致していなければなりません。

変数の順序とデータ型に矛盾がなければ、COMMON ブロックが同じサイズである必要はありません。チェイン先のプログラムで COMMON ブロックのサイズが小さい場合は余分な COMMON 変数が無視され、COMMON ブロックのサイズが大きい場合は残りの COMMON 変数が 0 または null 文字列で初期化されます。

参照　**OPTION EXPLICIT** ステートメント

CONST ステートメント

機能　数値や文字列の代わりに使う、記号定数を宣言します。

構文　**CONST** *constantname* = *expression*[, *constantname* = *expression*]...

constantname　記号定数の名前。記号定数名は、 Visual Basic の変数の名前付け規約に従います。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。型宣言文字が違っても、名前が同じなら同じ定数と見なされます。

expression　定数の値を示す式。文字や定数、べき乗 (^) を除く数値演算子や論理演算子からなる式を指定します。式の中で、文字列の結合, 変数, ユーザー定義関数, **SIN** や **CHR$** のような組み込み関数は使えません。

解説　型宣言文字を指定しても、その文字は記号定数名には含まれません。次のようになります。

```
CONST MAXDIM% = 250
.
.
.
DIM AccountNames$(MAXDIM)
```

型宣言文字を省略すると、*expression* に指定した式のデータ型と見なされます。数式の場合は、その式を計算した結果を表すことのできる、もっとも単純なデータ型になります。例えば式の計算結果が整数で表されるとき、定数は整数型と見なされます。

DEFINT などの **DEF***type* ステートメントは、記号定数のデータ型に影響しません。

記号定数は、参照される前に定義されていなければなりません。次の例では、記号定数 TWO より前に記号定数 ONE が定義されていないため、エラーとなります (定義はステートメントの左から右に向かって行われます)。

```
CONST TWO = ONE + ONE, ONE = 1
```

プロシージャの中で宣言された定数は、そのプロシージャの中だけで使えます。プロシージャの外で定義された定数は、モジュールの任意の場所で使えます。

次のような定数がよく使われます (0 以外の値は真を表します)。

```
TRUE = -1
```

記号定数は、値を変化させない変数を使うのに比べ、次のような利点があります。

- 一度定義するだけでモジュール全体で使える。
- 不注意で変更されない。
- コンパイルするときに、変数を使うときより効率的なコードが生成される。
- プログラムの修正が容易になる。

ControlBox プロパティ

機能

フォームの左上隅にコントロール メニュー ボックスを表示するかどうかを決めます。実行時には読み込み専用です。

構文

[*formname.*]**ControlBox**

対象

フォーム

解説

ControlBox プロパティは、デザイン時にフォーム デザイナで True (表示する) または False (表示しない) に設定できます。実行時には変更できません。デフォルトは True です。

コントロール メニューは、マウスを使わないユーザーがキーボードからウィンドウをコントロールする手段として便利です。また、コントロール メニュー ボックスをダブルクリックすると、アクティブ ウィンドウがクローズされます。最後のウィンドウがクローズされると、アプリケーション自身の実行が終了します。

ControlBox プロパティが True (表示する) に設定されていても、フォームの BorderStyle プロパティが 0 (なし) に設定されていると、コントロール メニュー ボックスは表示されません。

参照

BorderStyle プロパティ，MinButton プロパティ，MaxButton プロパティ

ControlPanel プロパティ

機能　画面各部の色など、システム全体の特性を決めます。

構文　**SCREEN.ControlPanel**(*subscript%*)[= *setting%*]

subscript%	ControlPanel 配列の添字 (0～17) を指定する数式。
setting%	各要素が示す項目に対する、色などの有効な設定値。

対象　SCREEN オブジェクト

解説　ControlPanel プロパティは、18 個の要素から構成されている配列です。それぞれの要素が 1 つのプロパティのようになっており、それぞれ取得および設定の対象が異なります。

各要素を指定するためには、カッコで囲んで添字 (*subscript%* 引数) を指定します。インクルード ファイル CONSTANT.BI では、この添字の指定に便利な定数が定義されています。これらの定数を使うためには、定数を使う各モジュールの先頭に、次のように **$INCLUDE** メタコマンドを置く必要があります。定数を使う代わりに、有効な値を持つ変数や数式で配列の要素を指定することもできます。

```
' $INCLUDE: 'CONSTANT.BI'
```

各要素の添字，その添字を示す定数，その要素の機能は次のとおりです。

0	ACCESSKEY_FORECOLOR	アクセス キーの文字色を決めます。0～15 の色番号を指定でき、デフォルトは 15 (明るい白) です。
1	ACTIVE_BORDER_BACKCOLOR	アクティブなウィンドウの境界の背景色を決めます。0～15 の色番号を指定でき、デフォルトは 7 (白) です。
2	ACTIVE_BORDER_FORECOLOR	アクティブなウィンドウの境界の文字色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。
3	ACTIVE_WINDOW_SHADOW	アクティブなフォームに影を付けるかどうかを決めます。True (-1) または False (0) が指定でき、デフォルトは True (-1) です。
4	COMBUTTON_FORECOLOR	コマンド ボタンの文字色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。この設定は、ControlPanel プロパティの THREE_D (添字 15) 要素が True (-1) に設定されているとき、高さが複数行のコマンド ボタンの境界色に対しては影響しません。
5	DESKTOP_BACKCOLOR	デスクトップの背景色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。
6	DESKTOP_FORECOLOR	デスクトップの文字色を決めます。0～15 の色番号を指定でき、デフォルトは 7 (白) です。

7	DESKTOP_PATTERN	デスクトップ上の背景として表示されるキャラクタを決めます。255 までの ASCII 文字を示す値が指定でき、デフォルトは 32 (スペース) です。0 (null 文字) を指定すると、そのとき表示されていた画面のイメージが保存され、バックグラウンドイメージとして表示されます。この状態でフォームを非表示にしても、前の画面イメージはそのまま残ります。また、出力ステートメントやグラフィック ステートメントは、イメージを上書きします。0 以外の値に変更すると、保存されていた画面イメージはクリアされます。
8	DISABLED_ITEM_FORECOLOR	無効に設定されている項目の文字色を決めます。0～15 の色番号を指定でき、デフォルトは 8 (灰色) です。
9	MENU_BACKCOLOR	メニューの背景色を決めます。0～15 の色番号を指定でき、デフォルトは 7 (白) です。
10	MENU_FORECOLOR	メニューの文字色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。
11	MENU_SELECTED_BACKCOLOR	選択 (反転表示) されているメニュー項目の背景色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。
12	MENU_SELECTED_FORECOLOR	選択 (反転表示) されているメニュー項目の文字色を決めます。0～15 の色番号を指定でき、デフォルトは 7 (白) です。
13	SCROLLBAR_BACKCOLOR	フォーム上に配置されている水平および垂直スクロール バーの背景色を決めます。0～15 の色番号を指定でき、デフォルトは 7 (白) です。この設定は、Attached プロパティが True (-1) に設定されている水平および垂直スクロール バーや、リストボックスのスクロール バーの色には影響しません。
14	SCROLLBAR_FORECOLOR	フォーム上に配置されている水平および垂直スクロール バーの前景色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。この設定は、Attached プロパティが True (-1) に設定されている水平および垂直スクロール バーや、リストボックスのスクロール バーの色には影響しません。
15	THREE_D	境界を持つコントロールを立体的に (3D で) 表示するかどうかを決めます。True (-1) または False (0) が指定でき、デフォルトは True (-1) です。True (-1) に設定すると、高さが複数行のコントロールは、上や左の境界の色と下や右の境界の色を分けることによって、立体的に見えるように表示されます。コマンド ボタンは手前に飛び出しているように表示され、それ以外のコントロールは奥に引っ込んでいるように表示されます。
16	TITLEBAR_BACKCOLOR	アクティブなフォームのタイトル バーの背景色を決めます。0～15 の色番号を指定でき、デフォルトは 0 (黒) です。

17	TITLEBAR_FORECOLOR	アクティブなフォームのタイトル バーの文字色を決めます。0～15 の色番号を指定でき、デフォルトは 15 (明るい白) です。

色を示す定数も、インクルード ファイル CONSTANT.BI で定義されています。

色番号は 0～15 の範囲ですが、実際に表示可能な色数はハードウェアに依存します。

参照　BackColor プロパティ，ForeColor プロパティ

COS, SIN, TAN 関数

機能　**COS** 関数は、引数に指定した数値のコサインを返す数値演算関数です。**SIN** 関数は、引数に指定した数値のサインを返す数値演算関数です。**TAN** 関数は、引数に指定した数値のタンジェントを返す数値演算関数です。

構文

COS(*numeric-expression*)

SIN(*numeric-expression*)

TAN(*numeric-expression*)

numeric-expression　ラジアン単位の角度示す数式。

解説

直角三角形の角のコサインは、その角に隣接している辺の長さと斜辺の長さとの比です。

直角三角形の角のサインは、その角の反対側の辺の長さと斜辺の長さとの比です。

直角三角形の角のタンジェント (**TAN** 関数が返す値) は、その角と反対側にある辺の長さとその角に隣接する辺の長さとの比です。

numeric-expression は、いずれの数値データ型でもかまいません。*numeric-expression* が整数または単精度の値であると単精度で計算します。その他の数値データ型であると、倍精度で計算します。

角度の単位を、度からラジアンに変換するには、度に π/180 (.0174532925199433) を掛けます。ラジアンを度に変換するには、ラジアンに 180/π (57.2957795130824) を掛けます。π は約 3.141593 です。

参照　**ATN** 関数

CSNG$ 関数

機能　文字列中の 2 バイト系文字を、1 バイト系文字に変換して返します。

構文　**CSNG$**(*string-expression$*)

string-expression$　変換元の文字列。

解説　*string-expression$* には、どんな文字列式を指定してもかまいません。**CSNG$** 関数は、指定された *string-expression$* 中の 2 バイト系文字を 1 バイト系文字に変換した結果を返します。文字列中に含まれた 1 バイト系文字と、変換対象外の 2 バイト系文字は変換されません。

この関数で変換される 2 バイト系文字は、すべての英数字，ひらがな，カタカナ，スペースと、以下の記号です。

、。，．　：；？！゛゜　｀　＾　￣　＿　－　／　｜　’　”　（）

［］　｛｝　「」　＋　－　＝　＜　＞　￥　＄　％　＃　＆　＊　＠

注　意　ひらがなは、すべて 1 バイト系のカタカナに変換されます。"バ"，"ぱ" などの濁音や半濁音は、ハ に変換します。また、ｷｴ などの対応する 1 バイト系文字がない文字は、発音の近いカタカナに変換されます。

参照　**CDBL$** 関数，**CSNG** 関数

CSRLIN 関数

POS 関数 を参照してください。

CtlName プロパティ

機能

コントロールを識別するための名前を決めます。実行時には取得および設定できません。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，テキスト ボックス，タイマー，垂直スクロール バー

解説

フォーム デザイナでコントロールを生成した直後のデフォルトの名前 (CtlName プロパティのデフォルト値) は、コントロールの種類に順に番号を付けたもの (Option1, Option2, Option3 など) になります。番号はコントロールの種類ごとに付けられ、1 から順に増えていきます。これにより、すべてのコントロールにはデフォルトでそれぞれ異なる名前が付けられます。

デフォルトのコントロール名は次のとおりです。最後の N は数字を示します。

コントロール	デフォルト名
オプション ボタン	OptionN
コマンド ボタン	CommandN
コンボ ボックス	ComboN
タイマー	TimerN
チェック ボックス	CheckN
ディレクトリ リスト ボックス	DirN
テキスト ボックス	TextN
ドライブ リスト ボックス	DriveN
ピクチャー ボックス	PictureN
ファイル リスト ボックス	FileN
フレーム	FrameN
メニュー	MenuN
ラベル	LabelN
リスト ボックス	ListN
垂直スクロール バー	VscrollN
水平スクロール バー	HScrollN

コントロールの名前を変更する場合は、他の名前と重複しないようにする必要があります。コントロール名は、アルファベットで始まり、アルファベット，数字，アンダースコア (_) だけで構成された、40 文字以内の文字列でなければなりません。また、"コントロール名" + "_" + "最も長いイベント名またはプロパティ名" が 40 文字以内になるように注意してください。

Visual Basic の予約語は、CtlName プロパティには設定できません。

CtlName プロパティと Caption プロパティを混同しないように注意してください。Caption プロパティを持つコントロールでは、デフォルトで 2 つのプロパティに同じ文字列が設定されますが、識別子として使われるのは CtlName プロパティです。

コントロール配列を使うときは、配列中のすべてのコントロールに対して、同じ CtlName プロパティを指定します。

プログラム中で CtlName プロパティ自身を参照することはできません。例えば次の行はエラーになります。

```
FOO$ = Command1.CtlName
```

コントロールを引数としてプロシージャに渡すときなど、どのコントロールであるかを判別する必要があるときは、CtlName プロパティではなく、Tag プロパティを使ってください。

コントロールの名前を変更しても、関連するイベント プロシージャの名前は自動的には変更されません。名前を変更したコントロールに対応させるには、イベント プロシージャの名前を変更する必要があります。

参照　FormName プロパティ，Index プロパティ，Tag プロパティ

Ctype 系関数 (CCUR, CINT, CLNG, CSNG, CDBL)

機能　指定された数式の値を、整数 (**CINT**)，長整数 (**CLNG**)，単精度浮動小数点数 (**CSNG**)，倍精度浮動小数点数 (**CDBL**)，通貨型数 (**CCUR**) にそれぞれ変換します。

構文

CINT(*numeric-expression*)

CLNG(*numeric-expression*)

CSNG(*numeric-expression*)

CDBL(*numeric-expression*)

CCUR(*numeric-expression*)

numeric-expression　任意の有効な数式。

解説　*numeric-expression* には、どんな精度の値を指定してもかまいません。それぞれの関数は、*numeric-expression* で指定した値を各データ型の変数に代入するのと同じ意味です。ただし *numeric-expression* の値が変換対象のデータ型で扱える範囲外であると、実行時に "オーバーフローしました" エラーが発生します。

CINT および **CLNG** 関数は、IEEE 規格のもっとも近い偶数への不偏丸めの規則に従って、小数点以下を丸めます。小数点以下が 0.5 以外のときは、通常の四捨五入と同じです。0.5 のときは、もっとも近い偶数に丸めます。通常の四捨五入の方法は、**INT** 関数を参照してください。

CSNG 関数は、必要に応じて変換前に値を丸めます。

より精度の高いデータ型に変換する場合の戻り値は、元の数式以上の精度を持つわけではないことに注意してください。変換によって付け加えられた桁は、元の数式が変換対象の精度で計算されているのでない限り、意味を持ちません。

参照　**INT** 関数，**CDBL$** 関数，**CSNG$** 関数

CURDIR$ 関数

機能　指定したドライブのデフォルトのディレクトリを示す文字列を返します。

構文　**CURDIR$**[(*drive$*)]

drive$　ドライブを指定する文字列式。最初の 1 文字だけが意味を持ちます。

解説　*drive$*で指定するドライブは、必ず存在していなければなりません。存在しない場合はエラーになります。指定できる範囲は、A から *lastdrive* までです。*lastdrive* は、CONFIG.SYS ファイルで設定可能な、最後のドライブ名です。

drive$ を省略するか、null 文字列 ("") を指定した場合は、カレント ディレクトリを返します。これは、OS のプロンプトに対してパスを指定せずに **CHDIR** コマンドを実行したときと似ています。

大文字小文字は区別されません ("C" は "c" と同じです)。

常に *drive$* に指定した最初の文字だけが使われるので、コロン (:) を付ける必要はありません。

CurrentX, CurrentY プロパティ

機能

現在の出力位置の水平位置または列座標 (CurrentX)、垂直位置または行座標 (CurrentY) を決めます。

構文

[*formname.*][*ctlname.*]**CurrentX**[= *coordinate%*]

[*formname.*][*ctlname.*]**CurrentY**[= *coordinate%*]

ctlname	ピクチャー ボックスの名前。
coordinate%	水平または垂直の位置を示す、0～254 の整数。デフォルトは 0 です。

対象

フォーム，ピクチャー ボックス

解説

CurrentX および CurrentY プロパティの値は、現在の出力位置、つまりフォームやピクチャー ボックスに対して **PRINT** メソッドを使ったときにテキストが表示される位置を示します。座標は、指定したフォームやピクチャー ボックスの左上を原点 (0, 0) として、キャラクタ単位で割り振られています。列座標および行座標と呼ぶこともあります。

次のメソッドを実行すると、CurrentX および CurrentY プロパティの値は自動的に変更されます。

メソッド	CurrentX, CurrentY の値
CLS	ともに 0 になります。
PRINT *exp*;	最後に出力された文字の直後の位置を示す値になります 。
PRINT *exp$*	CurrentY は 1 増え、CurrentX は 0 になります。
PRINT *exp$*,	次の出力領域の先頭を示す値になります。

PRINT メソッドの出力が、(CurrentX と CurrentY プロパティで示すことができる最大値である) 254 桁または行を超えた場合、エラーが発生します。

参照

CLS メソッド，**PRINT** メソッド

CV*x* 系関数 (CVC, CVI, CVL, CVS, CVD)

機能

FIELD ステートメントと組み合わせて、数値をランダムアクセス ファイルから読み込むために使う関数です。数値内部表現の文字列を、整数 (**CVI**)，長整数 (**CVL**)，単精度浮動小数点数 (**CVS**)，倍精度浮動小数点数 (**CVD**)，通貨型数 (**CVC**) にそれぞれ変換します。

構文

CVI(*string-expression$*)

CVL(*string-expression$*)

CVS(*string-expression$*)

CVD(*string-expression$*)

CVC(*string-expression$*)

string-expression$　任意の文字列式。

解説

これらの関数は、以前のバージョンと互換性を保つために残されています。ユーザー定義型変数を使うと、ランダムアクセス ファイルへの入出力が、より効率的にできます。

参照

CV*x*MBF 関数，**FIELD** ステートメント，**MK*x*$** 系関数

CV*x*MBF 系, MK*x*MBF$ 系関数 (CVSMBF, CVDMBF, MKSMBF$, MKDMBF$)

機能

CVSMBF 関数と **CVDMBF** 関数は、Microsoft バイナリ形式 (MBF) の数値を示す文字列を、IEEE 形式の数値に変換する関数です。**MKSMBF$** 関数と **MKDMBF$** 関数は、IEEE 形式の数値を、Microsoft バイナリ形式の数値を示す文字列に変換する関数です。

構文

CVSMBF(*4-byte-string*)

CVDMBF(*8-byte-string*)

MKSMBF$(*single-precision-expression!*)

MKDMBF$(*double-precision-expression#*)

4-byte-string	4 バイトの文字列。
8-byte-string	8 バイトの文字列。
single-precision-expression!	単精度浮動小数点数。
double-precision-expression#	倍精度浮動小数点数。

解説

これらの関数は、Microsoft バイナリ形式を使ったランダムアクセス ファイルに、実数を読み書きするために使います。Basic 以外の言語プログラムで作成したランダムアクセス ファイルを、Basic から利用するときに役立ちます。

CVSMBF 関数は、Microsoft バイナリ形式の数値を示す 4 バイトの文字列を、IEEE 形式の単精度浮動小数点数に変換します。**CVDMBF** 関数は、8 バイトの文字列を倍精度浮動小数点数に変換します。

MKSMBF$** 関数は、IEEE 形式の単精度浮動小数点数を、Microsoft バイナリ形式の数値を示す 4 バイトの文字列に変換します。**MKDMBF$ 関数は、倍精度浮動小数点数を 8 バイトの文字列に変換します。

実数データを Microsoft バイナリ形式でランダムアクセス ファイルに書き込むには、次の手順で行います。

- **MKSMBF$** (単精度) または **MKDMBF$** (倍精度) 関数を使って、数値を文字列に変換します。
- 変換結果を (**FIELD** ステートメントで定義された) 対応するフィールドに書き込みます。
- **PUT** ステートメントでレコードをファイルに書き込みます。

通貨型の数値は IEEE 浮動小数点形式では保存できません。したがって、通貨型の数値を Microsoft バイナリ形式の数値を表す文字列に変換することはできません。

参照

CV*x* 系関数，**MK***x***$** 系関数

DATA, READ, RESTORE ステートメント

機能

DATA ステートメントは、**READ** ステートメントで読み込むための、数値定数や文字列定数を定義します。**READ** ステートメントは、**DATA** ステートメントで定義された値を読み込み、変数に代入します。**RESTORE** ステートメントは、次にデータを読み出す **DATA** ステートメントを指定します。

構文

DATA *constant*[, *constant*]...

READ *variable*[, *variable*]...

RESTORE[{*label*|*linenumber*}]

constant	読み込まれるデータである数値定数または文字列定数。文字列がカンマ (,)，コロン(:)，前後のスペースを含むときは、文字列をダブル クォーテーション (") で囲む必要があります。
variable	読み込んだデータを代入する変数。
label	次に読み込む **DATA** ステートメントを示す行ラベル。
linenumber	次に読み込む **DATA** ステートメントを示す行番号。

解説

label と *linenumber* をどちらも省略すると、プログラムの最初の **DATA** ステートメントと見なされます。

DATA ステートメントは、プログラムの中でいくつでも定義できます。また **DATA** ステートメントには、1 行以内なら何個の定数を指定してもかまいません。各定数は、カンマ (,) で区切ります。

DATA ステートメントは、ソースファイルに書き込まれた順に処理されます。複数の **DATA**ステートメントに含まれる項目は、ステートメントに含まれる個数やプログラム中でステートメントが置かれている位置に関係なく、1 つの連続した項目として扱われます。

DATA ステートメントで、(**CONST** ステートメントで定義した) 記号定数を指定すると、定数の名前ではなく文字列と見なされます。次の例で 2 番目のデータ項目は、3.141593 という値ではなく、"PI" という文字列を表します。

```
CONST PI=3.141593          ' PI は定数 3.141593 として定義されます
.
.
.
DATA 2.20,PI,45,7          ' PI は文字列として扱われます
.
.
.
```

DATA ステートメントの行にコメントを記述する場合は、以下のようにコロン (:) が必要です。コロンを付けずに **REM** ステートメントを記述すると、"REM" という文字列と解釈されてしまいます。また ' も文字列と解釈されます。

```
DATA abc,123,45,edg    : ' コメントです
```

次のように空のデータ項目 (値を指定しない項目) も 1 つのデータになります。空のデータ項目は、数値変数に代入されると 0 になり、文字列変数に代入されると null 文字列 ("") になります。

```
DATA 1,2,,4,5
```

プログラミング環境での作業中、**DATA** ステートメントを記述できるのは、モジュール レベルだけです。Visual Basic は、ソース ファイルを読み込むとき、モジュール レベル以外の部分にあるすべての **DATA** ステートメントを、自動的にモジュール レベルに移します。

READ ステートメントは、**DATA** ステートメントと同じモジュールになければなりません。ただし、**READ** ステートメントはプロシージャ レベルにも置くことができます。

READ ステートメントは、**DATA** ステートメントから値を 1 つずつ読み出して、変数に順に代入します。モジュール内の **DATA** ステートメントで指定されたすべての値は連続したデータ項目として扱われます。1 つまたは複数の **READ** ステートメントで指定する変数の個数と、1 つまたは複数の **DATA** ステートメントで定義するデータの個数は、一致している必要はありません。データは順番に読み出されます。ただし、すべての **DATA** ステートメントで定義されているデータをすべて読み出したあとで、さ

らにデータを読み出そうとすると、エラーになります。逆に **READ** ステートメントで指定した変数の方が少なければ、余分なデータは無視されます。

数値変数に対して文字列を読み込もうとすると、エラーになります。また、その数値変数が表すことのできる最大値または最小値を超える値を読み込もうとすると、エラーになります。数値データの精度は、数値変数に合わせて代入されます。

文字列変数に対して数値を読み込もうとすると、値を表す数字が文字列として代入され、エラーは発生しません。固定長文字列変数に文字列を読み込むとき、変数より代入する文字列が長い場合は、超える部分は切り捨てられます。逆に短ければ、文字列は左詰めで格納され、残りの部分はスペースで埋められます。

ユーザー定義型の変数にデータを読み込むときは、各要素ごとに読み込んでください。

RESTORE ステートメントを使うと、以前に読み込まれた **DATA** ステートメントの値を再び読み込むことができます。

DATE$ 関数, DATE$ ステートメント

機能

DATE$** 関数は引数を取らず、現在の日付を表す文字列を返します。**DATE$ ステートメントは、現在の日付を設定します。

構文

**DATE$**[= *string-expression$*]

string-expression$ 設定する日付。

解説

日付は、次に示すいずれかの形式で表されます。

yy-mm-dd　yyyy-mm-dd　yy/mm/dd　yyyy/mm/dd

yyyy は年を表します (1980～2099)。mm は月を表します (01～12)。dd は日を表します (01～31)。

DATE$ ステートメントの引数で、1 桁の場合は先頭の 0 は省略できます。

ユーザーのコンピュータのシステム日付を **DATE$** ステートメントで変更すると、その変更はユーザーが再び変更するまで有効になります。

システム日付の設定に使用できる値の有効範囲は、シリアル番号を扱う関数で使われる有効範囲とは異なります。

参照

FORMAT$ 関数

DATESERIAL 関数

機能

数値で指定された日付を表す、シリアル値を返します。

構文

DATESERIAL(*year%*, *month%*, *day%*)

year%	年 (0～99 または 1753～2078)。
month%	月 (1～12)。
day%	日 (1～31)。

解説

引数 *year%* は、1900～1999 年の年号を指定する場合は、0～99 と指定してもかまいません。それ以外の年については、4 桁の年号 (例えば 1800) で指定してください。

通常の年月日の範囲外の引数は、有効な年月日を示すように評価されます。次の例では、引数 *month%* (13) は、次の年の最初の月として評価されます。したがってこの式は 1993 年 1 月 1 日を表します。

```
TheDate = DATESERIAL(92, 13, 1)
```

シリアル値の有効範囲 (-53,688～65,380) 外の値を返すような引数を指定すると、"引数が有効範囲にありません" エラーになります。この有効範囲 (-53,688～65,380) は、実際の日付に換算すると、1753 年 1 月 1 日～2078 年 12 月 31 日になります。

引数として負の数も指定できます。例えば、92 年 9 月 1 日の 1 週間前のシリアル値は、次のように求めることができます。

```
DATESERIAL(92, 9, 1-7)
```

DATEVALUE 関数

機能

文字列で指定された日付を表す、シリアル値を返します。

構文

DATEVALUE(*date$*)

date$	1753 年 1 月 1 日から 2078 年 12 月 31 日までの、任意の有効な日付を示す文字列。

解説

引数 *date$* には、"98/12/30/", "1998-Dec-28" などの書式で指定できます。

引数 *date$* に時刻のデータを含めても無視されます。次の例では、日付の指定だけが有効です。

```
DATEVALUE("98/12/30 2:24 AM")
```

日付の年号部分を省略すると、**DATEVALUE** 関数はそのとき使っているコンピュータのシステム時刻から現在の年を得て、その値を使います。

DAY 関数

機能　日付と時刻のシリアル値を評価して、1～31 の範囲で、月の何日目かを返します。

構文　**DAY**(*serial*)

serial　日付と時刻を表すシリアル値 (-53,688～65,380)。

DblClick イベント

機能　マウスをダブル クリックしたとき、つまりマウスに対して設定されているダブル クリック時間の内に、マウスの左ボタンを押して離し、再び押したときに発生します。

構文　**SUB Form_DblClick ()**

SUB *ctlname*_**DblClick** ([*Index* **AS INTEGER**])

Index　コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象　コンボ ボックス，ファイル リスト ボックス，フォーム，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス

解説　フォームの場合は、コントロールが配置されていない領域を、ユーザーがマウスの左ボタンでダブルクリックしたときに、DblClick イベントが発生します。

コントロールの場合は、次に示す操作が行われたときに、DblClick イベントが発生します。

- コントロールがマウスの左ボタンでダブルクリックされた。
- Style プロパティが 1 のコンボ ボックス，ファイル リスト ボックス，リスト ボックスの項目がダブルクリックされた。
- コード中から、ファイル リスト ボックスの FileName プロパティが、有効な存在するファイルの名前 (ワイルド カード文字を含まないファイル名) に設定された。

マウスの左右のボタンを区別するときは、MouseDown と MouseUp イベントを使います。

DblClick イベントは、1 つの操作で、複数の処理を行うような場合に便利です。例えば、リスト ボックスのある項目をダブルクリックすると、それが選択されると同時に、ダイアログ ボックスを閉じるような処理が行えます。

DblClick イベントが発生するときは、同時に MouseDown, MouseUp, Click イベントも発生します。発生する順番は、MouseDown, MouseUp, Click, DblClick, MouseUp です。

ユーザーがマウスの左ボタンを 2 回クリックしても、それがダブル クリック時間の設定時間内に行われなければ、オブジェクトは Click イベントが 2 回発生したと認識します。

参照　Click イベント

DECLARE ステートメント (Basic プロシージャ)

機能　Basic の SUB または FUNCTION プロシージャへの参照を宣言します。

構文

DECLARE FUNCTION *name*[([[**BYVAL**] *variable*[()][**AS** *type*]
[, [**BYVAL**] *variable*[()][**AS** *type*]]...])]

DECLARE SUB *name*[([[**BYVAL**] *variable*[()][**AS** *type*]
[, [**BYVAL**] *variable*[()][**AS** *type*]]...])]

name　プロシージャの名前。40 文字以内でなければなりません。FUNCTION プロシージャの名前の最後には、戻り値のデータ型を示す型宣言文字 (%, &, !, #, @, $) を付けることができます。

BYVAL　直後に指定されているパラメータが、参照による引き渡しではなく、値による引き渡しで渡されることを示します。ユーザー定義型の変数には使用できません。

variable　Visual Basic の変数名。配列を指定するときは、空のカッコを後ろに付けます。名前の最後に型宣言文字を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。

type　変数のデータ型。変数それぞれに対して指定します。 **INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, ユーザー定義型, **FORM**, **CONTROL** のいずれかが指定できます。 **FORM** は引数としてフォームが渡されることを示し、 **CONTROL** はいずれかのコントロールが渡されることを示します。

解説　プログラムが Basic で書かれているとき、**DECLARE** ステートメントは次のような場合に必要です。ただしプログラミング環境上では、プログラムがセーブされるときに、Visual Basic が自動的に **DECLARE** ステートメントを生成します。

- **CALL** ステートメントを使わずに SUB プロシージャを呼び出す場合
- 別のモジュールで定義されている FUNCTION プロシージャを呼び出す場合

DECLARE ステートメントは、必ずモジュール レベルに置かれていなければならず、プロシージャ内に置くことはできません。**DECLARE** ステートメントで行う宣言は、そのモジュール全体に対して有効です。

BYVAL キーワードは、その変数がデフォルトの引き渡し方法である参照による引き渡しではなく、値による引き渡しでプロシージャに渡されることを示します。**BYVAL** は、数値の基本データ型 (**INTEGER, LONG, SINGLE, DOUBLE, CURRENCY**) と可変長文字列 (**STRING**) に対してだけ指定できます。

参照による引き渡しでは、プロシージャ内でその引数の値を変更すると、呼び出し側の対応する変数の値も変更されます。値による引き渡しは、プロシージャ内でその引数の値を変更しても、呼び出し側の対応する変数の値には影響しません。

DECLARE ステートメントはまた、プロシージャを呼び出すときに使われる、引数の数とデータ型をチェックするように、コンパイラに指示します。

次のように、引数の指定方法によって、引数のチェックを行うかどうかが決まります。

宣言	意味
DECLARE SUB First	別のモジュールにあるプロシージャを宣言するときだけ、カッコ () を省略できます。この形式では、引数のチェックは行われません。
DECLARE SUB First ()	空のカッコだけを指定すると、そのプロシージャが引数を取らないことを示します。そのプロシージャを呼び出すときに引数を指定すると、エラーになります。
DECLARE SUB First (X AS LONG)	引数を指定すると、そのプロシージャが呼び出されるたびに、引数の個数とデータ型がチェックされます。

プロシージャに渡せる文字列は、可変長文字列だけです。**DECLARE** ステートメントの中で、固定長文字列を指定することはできません。ただし固定長文字列を要素として含むユーザー定義型の変数は指定できます。プロシージャを呼び出すステートメントの引数リストに固定長文字列を指定すると、プロシージャに引き渡される前に可変長文字列に変換されます。

配列変数全体をプロシージャに渡したいときは、名前の後ろに空のカッコを付けます。

```
DECLARE SUB DisplayText (A() AS STRING)
DIM Text$(100,5)
.
.
.
CALL DisplayText(Text$( ))
```

参照

CALL ステートメント (Basic プロシージャ), **DECLARE** ステートメント (Non-Basic プロシージャ), **FUNCTION** ステートメント, **SUB** ステートメント

DECLARE ステートメント (Non-Basic プロシージャ)

機能

Basic 以外の言語で書かれたプロシージャへの参照を宣言を宣言します。

構文

DECLARE FUNCTION *name*[**CDECL**][**ALIAS** "*aliasname*"][([[{**BYVAL**|**SEG**}] *variable*[**AS** *type*][, [{**BYVAL**|**SEG**}] *variable*[**AS** *type*]]...])]

DECLARE SUB *name*[**CDECL**][**ALIAS** "*aliasname*"][([[{**BYVAL**|**SEG**}] *variable*[**AS** *type*][, [{**BYVAL**|**SEG**}] *variable*[**AS** *type*]]...])]

FUNCTION　値を返す外部プロシージャを宣言することを示します。式の中から参照します。

SUB　値を返さない外部プロシージャ宣言することを示します。 Basic の SUB プロシージャと同じように呼び出します。

name　外部プロシージャの名前。 40 文字以内でなければなりません。値を返すプロシージャの名前の最後には、戻り値のデータ型を示す型宣言文字 (%, &, !, #, @, $) を付けることができます。

CDECL　そのプロシージャが C 言語の引数の順序，名前付け規約，呼び出し規約に従うことを示します。

ALIAS　そのプロシージャが .OBJ やライブラリ ファイルで別の名前を持つことを示します。

aliasname　外部ファイルやライブラリの中で使われている、プロシージャの別名。

BYVAL　直後に指定されているパラメータが、参照による引き渡しではなく、値による引き渡しで渡されることを示します。ユーザー定義型の変数には使用できません。

SEG　直後に指定されているパラメータが、セグメント アドレス (far アドレス) として引き渡されることを示します。

variable　Visual Basic の変数名。ここでは変数のデータ型だけが意味を持ちます。名前の最後に型宣言文字を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。

type　変数のデータ型。変数それぞれに対して指定します。 **INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, **ANY**, ユーザー定義型, **FORM**, **CONTROL** のいずれかを指定できます。

解説

Basic 以外の言語で書かれた外部プロシージャに対する呼び出しの方法を宣言します。この形式の **DECLARE** ステートメントを使うと、Basic 以外の言語で書かれたプロシージャを参照できます。**DECLARE** ステートメントはまた、プロシージャを呼び出すときに使われる、引数の数とデータ型をチェックするように、コンパイラに指示します。**DECLARE** ステートメントは、必ずモジュール レベルに置かれていなければならず、そのモジュール全体に対して有効です。宣言されたプロシージャは、**CALL** キーワードを使わずに呼び出すことができます。

Basic の規約では、引数は左から右の順で渡されますが、**CDECL** を指定すると右から左の順で引き渡します。**CDECL** は、オブジェクト ファイルとライブラリの検索に使用される名前にも影響します。

DECLARE ステートメントに **ALIAS** キーワードが無いと、そのプロシージャ名から型宣言文字を除いて先頭にアンダースコアを加えたものが、ライブラリと外部ファイルの検索時に使用される名前になります。

CDECL と **ALIAS** を共に指定すると、*aliasname* がそのまま使用されます。

BYVAL は、**INTEGER, LONG, SINGLE, DOUBLE, CURRENCY** 型にしか使えません。**BYVAL** をパラメータの前に指定すると、実引数は **DECLARE** ステートメントで指定したデータ型に変換された後で引き渡されます。特別なデータ型である **FORM** と **CONTROL** は、フォームまたはコントロールが引数として SUB または FUNCTION プロシージャに引き渡されるときにしか使えません。

Basic 以外の言語で書かれた外部プロシージャを宣言するときは、**AS** に続くデータ型の指定で **ANY** キーワードが使えます。**ANY** を指定すると、その引数の型チェックが行われません。引数を値によって引き渡す場合、**ANY** キーワードは指定できません。

BYVAL と **SEG** をどちらも指定しないと、引数は near アドレス (オフセット) として引き渡されます。

次のように、引数の指定方法によって、引数のチェックを行うかどうかが決まります。

宣言	意味
DECLARE SUB First CDECL	引数を指定しなければ、引数のチェックは行われません。
DECLARE SUB First CDECL( )	空のカッコだけを指定すると、そのプロシージャが引数を取らないことを示します。そのプロシージャを呼び出すときに引数を指定すると、エラーになります。
DECLARE SUB First CDECL(X AS LONG)	引数を指定すると、そのプロシージャが呼び出されるたびに、引数の個数とデータ型がチェックされます。

プロシージャに渡せる文字列は、可変長文字列だけです。**DECLARE** ステートメントの中で、固定長文字列を指定することはできません。プロシージャを呼び出すステートメントで実引数として固定長文字列を指定すると、プロシージャに引き渡される前に可変長文字列に変換されます。

Basic は、呼び出されたルーチンの実行が始まる前に、メモリ上の変数の位置を変えてしまうことがあります。このため、**SEG** キーワードを使ったり (キーワードなしで) near 参照によって渡すときは注意が必要です。**CALL** ステートメントの引数リストの中にメモリの移動を引き起こす要素を指定すると、問題の起こる可能性があります。単純変数や数式は near または far(SEG) 参照で安全に引き渡すことができます。配列に対して **SEG** は指定できません。

参照

CALL[S] ステートメント (Non-Basic プロシージャ), **DECLARE** ステートメント (Basic プロシージャ)

DEF FN ステートメント

機能

関数を定義し、名前を付けます。

構文

DEF FN*name*[(*parameterlist*)] = *expression*

DEF FN*name*[(*parameterlist*)]

[statementblock]

FN*name* = *expression*

[*statementblock*]

[**EXIT DEF**]

[*statementblock*]

END DEF

name	定義する関数の名前 (**DEF FN** 関数名)。必ず "FN" で始まります。モジュール中ですでに使われている名前は指定できません。**DEF FN** 関数名は、Visual Basic の変数の名前付け規約に従います。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。名前が同じでも、型宣言文字が違うと別の名前であると見なされます。
parameterlist	呼び出し側との受け渡しに使う変数とそのデータ型を、カンマで区切って指定します。
expression	定義する関数の戻り値を計算する式。ユーザー定義関数は、値を関数名に代入することで値を返します。
EXIT DEF	**DEF FN** ブロックを終了し、呼び出し側に値を返します。1 つの **DEF FN** ブロック中に複数の **EXIT DEF** を記述することもできます。

parameterlist 引数には、次の構文に従って引数とそのデータ型を指定します。

variable[()][**AS** *type*][, *variable*[()][**AS** *type*]]...

variable	渡された引数を受け取る変数の名前。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。**DEF FN** 関数では、引数として配列，ユーザー定義型，固定長文字列を使えません。引数は必ず値によって引き渡されます。
type	変数のデータ型。変数それぞれに対して指定します。**INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING** のいずれかが指定できます。

解説

DEF FN ステートメントは、ユーザー定義関数を定義します。ただし FUNCTION プロシージャの方がより柔軟な制御が可能であり、関数の定義には FUNCTION プロシージャを使うことをおすすめします。

DEF FN 関数は、定義されているモジュール内からしか呼び出せません。

DEF FN 関数は、呼び出す前に **DEF FN** ステートメントで定義されていなければなりません。定義する前に **DEF FN** 関数を呼び出すと、エラーになります。**DEF FN** 関数の定義の内部で、別の **DEF FN** 関数を定義することはできません。また、再帰的に呼び出せません。**DEF FN** 関数はその定義のあるモジュールの中でだけ有効で、他のモジュールから参照することはできません。

DEF FN 関数は、モジュール レベルの一部です。したがって変数はモジュールレベルコードと共有です。ただし、引数として渡された変数と、**DEF FN** ブロック内の **STATIC** ステートメントで宣言した変数は、ブロック内だけで有効なローカルな変数になります。引数として **DEF FN** ステートメントで指定した変数に対しては、必ず値で渡されます。

複数行の **DEF FN** 関数を途中で抜けるときは、**EXIT DEF** を使います。プログラムは、**DEF FN** 関数を呼び出した位置から実行が続けられます。

DEF FN 関数が返す値のデータ型は、次の条件によって決まります。

- 関数名 (**DEF FN** 関数名) の最後に型宣言文字 (%, &, !, #, @, $) が付けられていれば、それが示すデータ型。
- 指定されていなければ、**DEF***type* ステートメントで (*name* の) 先頭の文字が設定されてれば、そのデータ型。**DEF***type* ステートメントが使われていなければ、単精度浮動小数点 (**SINGLE**) 型。

関数名に対して値を代入しないと、戻り値として、数値関数では 0、文字列関数は null 文字列 ("") を返します。

DEF FN 関数は、数値か文字列を返します。*name* が文字列変数の名前であると、**DEF FN** 関数は文字列を返し、*name* が数値変数の名前であれば数値を返します。文字列関数に数値を割り当てたり、数値関数に文字列を割り当てたりすると、"型が一致しません" エラーになります。

戻り値が数値であるとき、**DEF FN***name* は *name* が示す精度を持つ値を返します。例えば、*name* が倍精度の値を示している場合は、**DEF FN***name* 関数が返す値は、*expression* の精度に関係なく倍精度の値になります。

Basic は、演算の効率を高めるため、数式を並べ替えることがあります。このため、変数の値を変えてしまうような **DEF FN** 関数は、数式の中で使わないでください。

入出力を行うステートメントから呼び出される **DEF FN** 関数では、入出力ステートメントを使わないでください。使用すると結果は予測できません。

参照

FUNCTION ステートメント，**SHARED** ステートメント，**STATIC** ステートメント，**DEF***type* ステートメント

DEF SEG ステートメント

機能

Absolute ルーチン，**BLOAD, BSAVE, POKE** ステートメント，**PEEK** 関数のために、セグメント アドレスを設定するステートメントです。

構文

DEF SEG[= *address*]

address　　セグメント アドレスの先頭を示す、0～65,535 の符号なし整数。

解説

address 引数には、カレント セグメントの新しいアドレスを示す 0～65535 の符号なし整数を指定します。範囲外の値を指定すると、"引数が有効範囲にありません" エラーが発生します。エラーが発生した場合は、それまでのセグメント値は変更されません。

セグメント アドレスの設定は、次に変更されるまでそのまま有効です。

引数を省略すると、カレント セグメントはデフォルトのデータ セグメント (DGROUP) に設定されます。

DEF と **SEG** は、必ずスペースで区切ってください。スペースで区切らないと、DEFSEG という名前の変数に対する代入と見なされてしまいます。

far メモリ領域に格納されているデータのアドレスにカレントのセグメント アドレスを設定するには、**SSEG** または **VARSEG** 関数を使うことができます (**SSEG** 関数は文字列データのカレント セグメント アドレスを返し、**VARSEG** 関数は数値データのカレント セグメント アドレスを返します)。例えば次のステートメントは、a$ という文字列が格納されているデータ セグメントに設定します。

```
DEF SEG = SSEG(a$)
```

DEF SEG は、EMS メモリ内の配列のセグメントの設定には使わないでください。/Ea オプションを指定して Visual Basic (VBDOS.EXE) を起動した場合、次の配列は EMS メモリに格納されることがあります。

- 512 バイト～ 16K バイトの数値配列
- 512 バイト～ 16K バイトの固定長文字配列
- 512 バイト～ 16K バイトのユーザー定義型配列

DEF SEG で配列のセグメントを設定したいときは、/Ea オプションを指定せずに Visual Basic (VBDOS.EXE) を起動してください。この場合、配列は EMS メモリに格納されません。

参照

Absolute ルーチン，**BLOAD** ステートメント，**BSAVE** ステートメント，**DEF FN** ステートメント，**PEEK** 関数，**POKE** ステートメント，**SSEG** 関数，**SSEGADD** 関数

Default プロパティ

機能

コマンド ボタンを、そのフォームのデフォルト ボタンにするかどうかを決めます。

構文

[*formname.*]*ctlname*.**Default**[= *boolean%*]

ctlname	コマンド ボタンの名前。
boolean%	デフォルト ボタンであるかどうかを示す値。 True (-1) がデフォルト ボタンを示し、False (0) がそうでないことを示します。デフォルトは False (0) です。

対象

コマンド ボタン

解説

デフォルト ボタンは、コマンド ボタン以外のコントロールがフォーカスを持っている場合に、[ENTER] キーが押されたときに Click イベントが発生するコマンド ボタンです。通常、最もよく選ばれると思われる処理を、デフォルト ボタンに設定しておきます。

同じフォーム上の、デフォルト ボタン以外のコマンド ボタンがフォーカスを持っている場合は、[ENTER] キーが押されると、(デフォルト ボタンではなく) フォーカスを持っているコマンド ボタンに対する Click イベントが発生します。

デフォルト ボタンに対して [ENTER] キーで Click イベントが発生したときは、そのときフォーカスを持っていたコントロールに対する KeyDown, KeyPress, KeyUp, Click などのイベントは発生しません。

1 つのフォーム上で、デフォルト ボタンに設定できる、つまり Default プロパティを True (-1) に設定できるコマンド ボタンは 1 つだけです。あるコマンド ボタンの Default プロパティを True (-1) に設定すると、他のすべてのボタンは自動的に False (0) に設定されます。

[ENTER] キーではなく、[ESC] キーが押されたときに Click イベントが発生するコマンド ボタンを設定するには、Cancel プロパティを使います。

参照

Cancel プロパティ, Click イベント

DEF*type* 系ステートメント

機能

変数およびユーザー定義関数 (**DEF FN** 関数，FUNCTION プロシージャ) のデフォルトのデータ型を設定します。

構文

DEFINT *letterrange*[, *letterrange*]...

DEFLNG *letterrange*[, *letterrange*]...

DEFSNG *letterrange*[, *letterrange*]...

DEFDBL *letterrange*[, *letterrange*]...

DEFCUR *letterrange*[, *letterrange*]...

DEFSTR *letterrange*[, *letterrange*]...

letterrange　デフォルトのデータ型を適用したい変数や関数の名前の範囲。*letter1*[-*letter2*] の書式で、変数名や関数名の最初の文字を指定します。*letter1* と *letter2* は任意の英字で、大文字でも小文字でもかまいません。*letter1* だけ指定すれば、その文字で始まる名前を持つ変数または関数が対象となります。*letter1-letter2* のように指定すると、その範囲全体が対象になります。

解説

これらのステートメントは、**AS** キーワードで明確にデータ型が宣言されている変数や、名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を示している変数や関数には影響しません。また、ユーザー定義型変数の要素には影響しません。

DEFINT は整数，**DEFLNG** は長整数，**DEFSNG** は単精度浮動小数点数，**DEFDBL** は倍精度浮動小数点数，**DEFCUR** は通貨型数，**DEFSTR** は文字列型を、それぞれデフォルトに設定します。

次の例では、Message は文字列変数となります。

```
DEFSTR A-Q
.
.
.
Message = "スタック領域が不足です．"
```

letterrange 引数の大文字と小文字は区別されません。次の 3 つのステートメントはすべて同じ意味です。

```
DEFINT I-N
DEFINT i-n
DEFINT i-N
```

プログラミング環境では、これらのいずれかのステートメントを使うと、以降に作成する各プロシージャの先頭に自動的に同じステートメントが挿入されます。

これらのステートメントが指定されていない場合、データ型が宣言されていない変数および関数は単精度型になります。

例えば、変数 I%, I&, I!, I#, I@, I$ は、すべて別の変数として扱われ、それぞれ異なる値を格納できるということに注意してください。

DIM ステートメント

機能

変数を宣言します。また、変数のデータ型を決めます。

構文

DIM[**SHARED**] *variable*[([*subscripts*])][**AS** *type*][, *variable*[([*subscripts*])][**AS** *type*]]...

SHARED	宣言された変数をモジュール内のすべてのプロシージャで共有することを示します。変数は、そのモジュールに対してグローバルになります。プロシージャの中で、個々に **SHARED** ステートメントを使う必要はありません。**SHARED** を指定できるのは、モジュールレベル コード内だけです。
variable	宣言する変数の名前。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。
subscripts	各次元の添字の最大値と最小値。これにより、配列の次元数も決まります。次元数の最大値は 60 です。
type	変数のデータ型 (**INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, **STRING** * *n* またはユーザー定義型)。変数それぞれに対して指定します。省略すると、変数名に型宣言文字が付けられていればそのデータ型になり、付けられていなければ **SINGLE** (単精度浮動小数点数) と見なされます。

解説

宣言の方法

subscripts 引数には、次の構文に従い、配列変数の次元と、各次元の添字の下限と上限を指定します。

[*lower* **TO**]*upper*[, [*lower* **TO**]*upper*]...

lower は添字の最小値を示し、*upper* は最大値を示します。*lower* のデフォルトは 0 ですが、**OPTION BASE** ステートメントを使うと、デフォルトを 1 に設定できます。デフォルトが 0 のとき、次の宣言はすべて同じ意味です。

```
DIM A(8, 3)
DIM A(0 TO 8, 0 TO 3)
DIM A(8, 0 TO 3)
```

添字に負の数を指定することもできます。指定できる範囲は -32768～32767 です。**DIM** または **COMMON** ステートメントで宣言されていない配列を使う場合、配列の添字の最大値は 10 です。

```
DIM A(-4, TO 10)
DIM B(-99 TO -5, -3 TO 0)
```

数値配列が占めるメモリのサイズは、要素数と、その配列のデータ型の格納に必要なバイト数の積として計算できます。例えば、配列全体のサイズが同じなら、整数型 (2 バイト) の配列は、長整数型 (4 バイト) の配列の 2 倍の要素を持つことになります。文字列配列のサイズは、1 つの配列変数につき全体で最大約 64K バイトに制限されています。

配列変数の要素を参照するとき、指定された範囲外の値を添字として指定すると、エラーが発生します。配列のサイズが制限を超える場合もエラーになります。また、配列を参照したあとで、その配列を **DIM** ステートメントで宣言しようとしてもエラーが発生します。/D オプションを指定せずにコンパイルしたプログラムでは、添字の範囲のチェックは行われません。

DIM ステートメントは、数値配列および文字配列のすべての要素を初期化します。初期化された結果、各要素は数値配列では 0、可変長文字列配列は null 文字列になります。また、ユーザー定義型の各要素も、固定長文字列のフィールドも含めて、同様に初期化されます。固定長文字列は、スペースではなく CHR$(0) で初期化されます。

DIM ステートメントで **SHARED** キーワードを指定すると、モジュール内のすべてのプロシージャで変数が共有されます。これに対して、**SHARED** ステートメントでは 1 つのプロシージャとモジュールレベル コードの間で変数を共有するために使います。

静的 (STATIC) 配列と動的 (DYNAMIC) 配列

配列の宣言の方法によって、その配列が、コンパイル時にメモリを割り当てられる静的 (STATIC) 配列か、実行時にメモリを割り当てられる動的 (DYNAMIC) 配列かを決めることができます。静的か動的かは、**$STATIC** および **$DYNAMIC** メタコマンドによっても影響を受けます。

先に **COMMON** ステートメントで宣言された配列は、静的配列になります。

明確に宣言されずに、コード モジュールのモジュール レベルで使われた配列は、動的配列になります。

数値定数や記号定数を使って、添字の最小値と最大値が指定された配列は、**$STATIC** メタコマンドが使われていない限り、静的配列になります。

変数を使って、添字の最小値と最大値が指定された配列は、動的配列になります。

各次元の添字の上限と下限を指定せずに (空のカッコを指定して) **DIM** ステートメントを使うと、その配列は動的配列になります。配列の次元と各次元の添字の上限，下限は、プロシージャ レベルで **REDIM** ステートメントを使って定義してください。

型宣言

DIM ステートメントは、変数のデータ型を宣言するためにも使います。この場合、変数名の後に型宣言文字 (%, &, !, #, @, $) を付けるか、**AS** キーワードの後に **INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING** のいずれかを指定します。固定長文字列を宣言するには **STRING** **n* (*n* はバイト数) のように指定してください。また、ユーザー定義型の変数を宣言することもできます。次の例では、Card というユーザー定義型がモジュール レベルで定義されています。この型の変数 TopCard は、モジュール レベルまたはプロシージャ中で **DIM** ステートメントを使って宣言されます。

```
TYPE Card                    ' モジュール レベル
    Suit AS STRING * 9
    Value AS INTEGER
END TYPE
DIM TopCard As Card          ' モジュールまたはプロシージャ レベル
```

ユーザー定義型の配列を宣言することもできます。

```
DIM Deck(1 TO 52) AS Card
```

参照　**REDIM** ステートメント，**COMMON** ステートメント，**OPTION EXPLICIT** ステートメント

DIR$ 関数

機能　指定した文字列パターンとマッチするファイル名を返します。

構文　**DIR$**[(*filespec$*)]

*filespec$*　ファイル名を調べたいドライブやディレクトリ、または特定のファイルを示す文字列式。疑問符 (?) やアスタリスク (*) などの MS-DOS のワイルドカード文字も使えます。省略すると、直前の **DIR$** 関数と同じと見なされます。

解説　*filespec$* 引数には、MS-DOS の DIR コマンドに対する指定と同じように指定します。文字列変数か文字列定数で指定してください。**DIR$** は、指定された *filespec$* とマッチする最初のファイル名を返します。このとき返るのはファイル名だけで、ドライブやディレクトリの指定は含まれません。

*filespec$* とマッチする次のファイル名を得るためには、引数を指定せずに再び **DIR$** を呼び出してください。ファイル名が取り出される順序は、MS-DOS の DIR コマンドと同じです。マッチするファイル名が無いときは、**DIR$** は null 文字列を返します。新しい *filespec$* で再び **DIR$** を使う前に、前の *filespec$* とマッチするすべてのファイル名を取り出す必要はありません。

**DIR$** 関数を最初に呼び出すときに *filespec$* を指定しないと、"引数が有効範囲にありません" エラーが発生します。

大文字小文字は区別されません ("C" は "c" と同じです)。

ディレクトリ名や Hidden 属性を持つファイルは対象になりません。

DO...LOOP ステートメント

機能

指定した条件が真である間、または指定した条件が真になるまで、一連のステートメントの実行を繰り返すフロー制御ステートメントです。

構文

ループの前で条件判断をする場合

DO[{**WHILE**|**UNTIL**} *condition*]

[*statementblock*]

[**EXIT DO**]

[*statementblock*]

LOOP

ループの後で条件判断をする場合

DO

[*statementblock*]

[**EXIT DO**]

[*statementblock*]

LOOP[{**WHILE**|**UNTIL**} *condition*]

condition	ループの終了または続行条件を示す、真 (0 以外の値) または偽 (0) を返す式。
statementblock	ループ内で実行される、任意の行数のステートメント。
EXIT DO	ループを終了し、**LOOP** キーワードの次のステートメントに移ることを示します。

解説

WHILE キーワードを指定した場合は、*condition* にはループの続行条件を指定します. *condition* が真の間は、ループ内のステートメント (*statementblock*) が繰り返し実行されます。**UNTIL** キーワードを指定した場合は、*condition* にはループの終了条件を指定します。*condition* が真になると、ループの実行が終了します。

LOOP に対して **WHILE** または **UNTIL** を指定した場合、ループは最低 1 回実行されます。

ループ内で **EXIT DO** を使うと、**WHILE** または **UNTIL** で指定した以外の条件で、ループを終了させることができます。**EXIT DO** は、**DO...LOOP** ループの中でのみ使用できます。

DO...LOOP ループはネストできます。つまり、**DO...LOOP** ループの内部に別の **DO...LOOP** ループを入れることができます。この場合、**LOOP** は最後に実行された **DO** に対応します。**DO** と **LOOP** が 1 対 1 に対応していないと、"LOOP がありません" または "DO がありません" エラーが発生します。ループ内で **EXIT DO** が実行されると、その **EXIT DO** を囲んでいる 1 番内側のループから抜け出します。

DO を実行せずに **DO...LOOP** ループ中に制御を移すことは避けてください。原因の突き止めにくい、実行時エラーやプログラムの論理的なエラーを引き起こします。

DO...LOOP ステートメントは、ループの先頭または最後でループ条件をテストできるため、**WHILE...WEND** や **FOR...NEXT** ステートメントより柔軟な処理ができます。

参照　**FOR...NEXT** ステートメント，**WHILE...WEND** ステートメント

DOEVENTS 関数

機能　すでにイベントが発生していないかどうかをチェックし、そのときロードされているフォームの数を返します。引数は取りません。

構文　**DOEVENTS()**

解説　通常は、イベント プロシージャの実行中はユーザーの操作は受け付けられず、イベントは処理されません (これはタイマー イベントであっても同じです)。例えば、大量のデータをソートするなどの長い時間がかかる処理をイベント プロシージャで行おうとするとき、取り消しを意味するコマンド ボタンを用意しても意味がないことになります。処理に長時間かかるイベント プロシージャを使う場合、イベント プロシージャ中で **DOEVENTS** 関数を呼び出すと、イベントが発生していないかどうかを調べることができます。もし発生したら、そのイベントはすぐにオブジェクトによって認識され、対応するイベント プロシージャに制御が移ります。発生していなければ、前のイベント プロシージャの実行がそのまま続けられます。また、**DOEVENTS** 関数を使うことによって別のイベント プロシージャが呼び出された場合、そのイベント プロシージャの実行が終了すると、前の (**DOEVENTS** 関数を含む) イベント プロシージャに制御が戻ります。

DOEVENTS 関数は、そのときロードされているフォームの数を返します。

DOEVENTS 関数は、イベント プロシージャ以外でも使うことができます。

DOEVENTS 関数を使うときには注意が必要です。例えば、コマンド ボタンのクリックによってあるイベント プロシージャが実行され、その中でファイルをオープンして処理する場合を考えてみます。この場合、そのイベント プロシージャの実行中に、ユーザーがコマンド ボタンをもう一度クリックしてしまう可能性があります。この状態で **DOEVENTS** 関数が呼ばれると、再び同じイベント プロシージャの実行が開始され、すでにオープンされているファイルをオープンしようとして、エラーが発生してしまいます。

DOEVENTS 関数を使うときは、グローバル変数にも注意してください。**DOEVENTS** 関数の呼び出しによって実行されたイベント プロシージャによって、元のプロシージャが使っているグローバル変数の値が書き換えられてしまう可能性があります。

DRAG メソッド

機能　コントロールのドラッグの開始，終了，キャンセルを行います。

構文　[{[*formname.*]*ctlname.*|*formname.*}]**DRAG** [*action%*]

ctlname	ドラッグするコントロールの名前。
action%	実行される動作を示す整数値。0 がキャンセル，1 が開始，2 が終了を示します。

対象　チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フレーム，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説　*action%* 引数に指定する値によって、行う操作を指示します。0 がドラッグ & ドロップ操作のキャンセル，1 が開始，2 が終了 (コントロールのドロップ) を示します。*action%* 引数を省略すると、開始 (1) と見なされます。

DRAG メソッドは、コントロールのドラッグ & ドロップ操作を制御するために使います。DragMode プロパティが 0 (マニュアル) に設定されているコントロールをドラッグするには, 必ず **DRAG** メソッドが必要です. ただし DragMode プロパティが 1 (自動) であるコントロールに対しても使うことができます。

コントロールがドラッグされている間のマウス カーソルの形を変えたいときは、MousePointer プロパティを使ってください。

参照　DragDrop イベント，DragMode プロパティ，DragOver イベント，MousePointer プロパティ

DragDrop イベント

機能　ドラッグ & ドロップ操作が完了したときに発生します。

構文　**SUB Form_DragDrop** (*Source* **AS CONTROL**,*X* **AS SINGLE**, *Y* **AS SINGLE**)

SUB *ctlname*_**DragDrop** ([*Index* **AS INTEGER**,]*Source* **AS CONTROL**,
X **AS SINGLE**, *Y* **AS SINGLE**)

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

Source ドラッグされた元 (ソース) のコントロールの名前が渡されます。

X, Y ドラッグ先 (ターゲット) のフォームまたはコントロールにおける、マウス ポインタの水平位置 (X) および垂直位置 (Y) が渡されます。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

あるコントロールが別のコントロールまたはフォーム上までドラッグされ、マウス ボタンが離されるか、action% = 2 で **DRAG** メソッドが使われると、ドラッグ & ドロップ操作が完了します。このとき DragDrop イベントが発生します。

DragDrop イベントに対するイベント プロシージャには、ドラッグ & ドロップ操作が完了した後の動作を記述します。例えば、ドラッグされたコントロールを新しい位置に移動したり、ファイルをコピーしたりする処理に利用できます。

Source 引数を使うと、ドラッグされたコントロールのプロパティを参照できます。また、このコントロールにメソッドを使うこともできます。

```
Source.Visible = 0
```

何種類かのコントロールがドラッグされる可能性があるときは、**IF** ステートメントと **TYPEOF** キーワードを使って、そのコントロールの種類を調べます。また、Tag プロパティをあらかじめ設定しておくと、同じ種類のコントロールが複数ある場合でも、1 つのコントロールを特定することができます。

ドラッグの開始方法を指定するには、DragMode プロパティと **DRAG** メソッドを使ってください。また、ドラッグが開始されてから DragDrop イベントが発生するまでのイベントは、DragOver イベントを使って処理できます。

参照

DRAG メソッド，DragMode プロパティ，DragOver イベント

DragMode プロパティ

機能

ドラッグ & ドロップ操作のモードを決めます。

構文

[*formname*.]*ctlname*.**DragMode**[= *mode%*]

mode%　0 (マニュアル) または 1 (自動)。デフォルトは 0 です。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フレーム，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

DragMode プロパティは、ドラッグ & ドロップ操作を行うときのモードを決めます。"ドラッグ & ドロップ" とは、ユーザーがマウスを使ってコントロールをドラッグし、それをフォームや他のコントロールの上にドロップできる機能を表します。DragMode プロパティが 0 (マニュアル) に設定されているときは、ユーザーがコントロールをドラッグしようとしても、コントロールは動きません。ドラッグするためには、コード中で **DRAG** メソッドを使う必要があります。1 (自動) に設定されているときは、ユーザーがコントロールをドラッグしようとすると、自動的にドラッグが開始されます。

DragMode プロパティを 1 (自動) に設定すると、そのコントロールは Click などの通常のイベントを認識しなくなりますから注意してください。したがって通常は、DragMode プロパティは 0 (マニュアル) に設定しておきます。

ドラッグ & ドロップ操作を行うときは、マウスやキーボードに関連するイベント (KeyDown, KeyPress, KeyUp, MouseDown, MouseMove, MouseUp) と **DRAG** メソッドを組み合わせて、コード中から動作を制御してください。DragDrop イベントがドラッグ & ドロップ操作の終わりを意味します。また、DragOver イベントを使うと、ドラッグされているコントロールと別のコントロールとの位置関係を知ることができます。

コントロールがドラッグされている間、ユーザーの操作によって発生する、マウスやキーボードに関連した他のイベント (KeyDown, KeyPress, KeyUp, MouseDown, MouseMove, MouseUp) は、コントロールによって認識されません。ただし、コントロールはコード中からの操作によって発生したイベントは認識します。

参照

DRAG メソッド，DragDrop イベント，DragOver イベント

DragOver イベント

機能

ドラッグ & ドロップ操作が進行しているときに発生します。

構文

SUB Form_DragOver (*Source* **AS CONTROL,**
X **AS SINGLE,** *Y* **AS SINGLE,** *State* **ASINTEGER)**

SUB *ctlname*_**DragOver** ([*Index* **AS INTEGER,**] *Source* **AS CONTROL,**
X **AS SINGLE,** *Y* **AS SINGLE,** *State* **AS INTEGER)**

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

Source ドラッグされた元 (ソース) のコントロールが渡されます。

X, Y ドラッグ先 (ターゲット) のフォームまたはコントロールにおける、マウス ポインタの水平位置 (X) および垂直位置 (Y) が渡されます。

State ドラッグされているコントロールが、ターゲットのフォームまたはコントロールに対して、どのような状態にあるかを示す値が渡されます。

対象

チェック ボックス, コンボ ボックス, コマンド ボタン, ディレクトリ リスト ボックス, ドライブ リスト ボックス, ファイル リスト ボックス, フォーム, フレーム, 水平スクロール バー, ラベル, リスト ボックス, オプション ボタン, ピクチャー ボックス, テキスト ボックス, 垂直スクロール バー

解説

DragOver イベントは、ドラッグの開始からターゲット上にドロップされるまでの間に、ソースのコントロールがどのような状態にあるかを調べるために使います。

State 引数には、ソースのコントロール (マウス ポインタ) とターゲットの位置関係を示す、次のような値が渡されます。

値	意味
0	範囲内に入る: ソースのコントロールはターゲットの範囲の中へ入るところです。
1	範囲外に出る: ソースのコントロールはターゲットの範囲の外へ出るところです。
2	範囲の上: ソースのコントロールはターゲット上を移動しています。

DragOver イベントは、そのときマウス ポインタが位置しているオブジェクトによって認識されます。

ドラッグされているコントロールの名前は、*Source* 引数で調べられます。このコントロールにメソッドを使うこともできます。

```
Source.Visible = 0
```

DragOver イベントを利用すると、次のような処理が可能です。

- ターゲットの領域に入っていることを示すために、ForeColor や BackColor プロパティを使ってターゲットとなるオブジェクトの色を変更する。
- ドラッグ中であることを示すために、MousePointer プロパティを使ってマウス ポインタの形を変更する。

ドラッグの開始方法を指定するには、DragMode プロパティと **DRAG** メソッドを使ってください。

参照　**DRAG** メソッド，DragDrop イベント，DragMode プロパティ

DRAW ステートメント

機能　指定された文字列式を解釈し、それに従って図形を描きます。フォームが表示されている状態では利用できません。

構文　**DRAW** *string-expression$*

string-expression$　グラフィックスを描くためのマクロ コマンド。

解説　フォームが表示されているときに **DRAW** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **DRAW** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

DRAW ステートメントは、他のグラフィックス ステートメントの機能を 1 つのグラフィックス マクロ言語にまとめたものです。このマクロ言語は、あるイメージを描画するための一連の記述を定義するものです。これらの記述には、移動 (上，下，左，右)，色指定，回転，拡大/縮小などが含まれます。引数 *string-expression$* は、これらのマクロ コマンドからなります。

マクロコマンドには、次の 3 種類のものがあります。

- 移動コマンド
- 回転，色，拡大/縮小コマンド
- サブストリング コマンド

移動コマンド

移動コマンドは、指定した方向に指定した距離だけ、グラフィックス出力位置を移動するためのコマンドです。コマンドの前に "B" を付けなければ、そこに線が引かれます。それぞれの移動コマンドは、現在のグラフィックス出力位置を起点にして移動を行います。現在のグラフィックス出力位置とは、最後に実行したグラフィックスコマ

ンドで、描画が終了した点を示します。プログラムの起動直後などのデフォルトのグラフィックス出力位置は、画面の中央です。

移動距離の単位は、S コマンドを実行していないデフォルトの状態では、ポイント (ピクセル) 単位です。移動距離の指定を省略すると、1 単位と見なされます。

コマンド	機能
[B]	他のコマンドの前に付け、線を引かずに移動することを指定します。
[N]	他のコマンドの前に付け、移動後に元の位置に戻ることを指定します。
U[*n*]	*n* 単位だけ、上に移動します。
D[*n*]	*n* 単位だけ、下に移動します。
L[*n*]	*n* 単位だけ、左に移動します。
R[*n*]	*n* 単位だけ、右に移動します。
E[*n*]	上に *n* 単位、右に *n* 単位、斜めに移動します。
F[*n*]	下に *n* 単位、右に *n* 単位、斜めに移動します。
G[*n*]	下に *n* 単位、左に *n* 単位、斜めに移動します。
H[*n*]	上に *n* 単位、左に *n* 単位、斜めに移動します。
M[{+|-}]*x*,*y*	*x* と *y* で指定した位置まで移動します。符号を指定した場合は、*x* と *y* は現在のグラフィックス出力位置に対する相対座標を示します。符号を指定していない場合は、絶対座標を示します。

回転、色指定、拡大/縮小コマンド

次のコマンドは、回転方向や色，拡大/縮小率などを指定し、描画に変化を付けるために使います。

コマンド	機能
A*n*	回転角度を設定します。*n* には 0～3 の整数を指定し、0 は 0 度，1 は 90 度，2 は180 度，3 は270 度を示します。90 度または 270 度回転させた図形は、縦横比に合わせて調整されるため、縦横比 3:4 の標準画面上でも、0 度または 180 度回転させたときの図形と同じサイズで表示されます。

TA*n*	回転角度を *n* 度に設定します。以降描く図形は、反時計回りに *n* 度回転した状態で描かれます。*n* には、-360～360 の整数を指定します。*n* に負の値を指定すると、時計回りに回転させたことと同じ意味になります。次の例は TA コマンドを使って、10 度ずつずらした円の半径を描いています。
	`FOR D=0 TO 360 STEP 10`
	`DRAW "TA="+VARPTR$(D)+"NU50"`
	`NEXT D`

C*n*	図形の色を *n* に設定します。色番号，パレット番号の詳細は、**COLOR, PALETTE, SCREEN** ステートメントを参照してください。

S*n*	拡大/縮小率を *n* に設定します。*n* には、1～255 の整数を指定します。実際にカーソルが移動する距離は、 U, D, L, R, M コマンドで指定した距離に、ここで設定した拡大 /縮小率をかけた距離になります。
P*p,b*	図形の内部を塗りつぶす色を *p* に、境界線の色を *b* に設定します。**DRAW** ステートメントでは、塗りつぶしパターンを使った塗りつぶしはできません。

サブストリングコマンド

コマンド	機能
X*string-expression*$	サブストリングを実行します。このコマンドを使って、あるマクロ文字列から別のマクロ文字列を実行することができます。 1 つのマクロ文字列から別のマクロ文字列を実行し、このマクロ文字列から、さらに別のマクロ文字列を実行するというように、次々にマクロ文字列を実行することができます。 数値引数として、123 のような定数や変数名を使うことができます。 Basic でサブストリングコマンドを実行するには、次の構文を使います。 "X" + VARPTR$(*string-expression$*)

参照

SCREEN ステートメント，**LINE** ステートメント，**PSET** ステートメント，**COLOR** ステートメント，**PALETTE** ステートメント

Drive プロパティ

機能

ドライブ リスト ボックスでそのとき選択されているドライブの取得と設定を行います。デザイン時には無効です。

構文

[*formname*.]*ctlname*.**Drive**[= *drivespec$*]

ctlname	ドライブ リスト ボックスの名前。
drivespec$	ドライブの名前を示す文字列式。そのとき有効なドライブ名でなければなりません。デフォルトは、コントロールがロードされた時点でのカレント ドライブです。

対象

ドライブ リスト ボックス

解説

Drive プロパティを使って現在設定されているドライブ名を調べる場合、"A:" の形式で返されます。また、そのドライブにボリューム ラベルが設定されていたら、"A: [MS-DOS_5] のような形式で返されます。

Drive プロパティを使ってドライブを変更する場合、文字列の先頭の 1 文字だけが有効です。コロンは省略できます。大文字小文字は区別されません。存在しないドライブを指定すると、エラーが発生します。また、フロッピー ディスクがセットされていないフロッピー ディスク ドライブを指定すると、“ディスクが準備されていません” エラーが発生します。

ユーザーのドライブ リスト ボックスに対する操作またはコード中からの操作によって Drive プロパティの設定が変更されると、Change イベントが発生します。

Drive プロパティの設定が変更されると、そのとき有効なドライブが自動的に再度チェックされます。これによって、実行中に接続されたネットワーク上のドライブを調べることができます。

そのとき選択 (反転表示) されている項目は、List プロパティと ListIndex プロパティの組み合わせでも取得できます。次の 2 つは同じ意味です。

ctlname.Drive

ctlname.List(*ctlname*.ListIndex)

参照　ディレクトリ リスト ボックス コントロール，ファイル リスト ボックス コントロール，Path プロパティ

DropDown イベント

機能　コンボ ボックスのリスト部分がドロップ ダウンするときに発生します。Style プロパティが 1 (シンプル コンボ ボックス) の場合は発生しません。

構文　**SUB** *ctlname*_**DropDown** ([*Index* **AS INTEGER**])

Index　コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象　コンボ ボックス

解説　DropDown イベントに対するイベント プロシージャは、ユーザーが選択を行う前に、コンボ ボックスのリスト部分の内容を変更するために使います。このプロシージャでは、**ADDITEM** または **REMOVEITEM** メソッドを使って、リストに項目を追加したり削除したりできます。この方法は、複数のコントロールに相関性を持たせたいときに便利です。例えば、オプション ボタンの選択状態に従って、コンボ ボックスのリスト部分に表示する内容を変化させることができます。

参照　**ADDITEM** メソッド，**REMOVEITEM** メソッド，Style プロパティ

$DYNAMIC, $STATIC メタコマンド

機能

配列をメモリに割り当てる方法を指定するメタコマンドです。**$STATIC** メタコマンドは、そのモジュール中の配列をデフォルトで静的配列にすることを示し、**$DYNAMIC** メタコマンドは動的配列にすることを示します。

構文

{'|**REM**} **$DYNAMIC**

{'|**REM**} **$STATIC**

$DYNAMIC	そのモジュール中の配列を、デフォルトで動的配列にすることを示します。配列に対するメモリ割り当ては、プログラム実行時に決まります。
$STATIC	そのモジュール中の配列を、デフォルトで静的配列にすることを示します。配列に対するメモリ割り当ては、コンパイル時に決まります。

解説

すべてのメタコマンドは、行頭にシングル クォーテーション (') または **REM** を付け、コメント行として記述します。

配列が動的または静的であることを指定するには、通常は **DIM** や **REDIM** ステートメントを使う方が便利です。

$STATIC メタコマンドは、そのモジュール中の配列を、デフォルトで静的配列にすることを示します。配列に対するメモリ割り当ては、コンパイル時に決まります。静的配列は、**ERASE** ステートメントで (0 または null 文字列に) 初期化することができますが、割り当てられているメモリを解放したり、**REDIM** ステートメントでメモリ サイズを変更することはできません。

$DYNAMIC メタコマンドは、そのモジュール中の配列を、デフォルトで動的配列にすることを示します。配列に対するメモリ割り当ては、プログラム実行時に決まります。動的配列に対して **ERASE** ステートメントを実行すると、その配列は消去され、割り当てられていたメモリは解放されます。また、**REDIM** ステートメントを実行すると、配列のサイズを変更することができます。

$STATIC および **$DYNAMIC** メタコマンドは、暗黙に宣言された配列 (**DIM** ステートメントなどで宣言されずにモジュール レベルで使われた配列) を除いたすべての配列に影響を与えます。ただし、**$STATIC** および **$DYNAMIC** メタコマンドにかかわらず、宣言の順番など、別の条件で静的か動的かが決まる場合もあります。暗黙に宣言された配列は、常に静的配列になります。

プロシージャの内部の配列は、**SUB** または **FUNCTION** ステートメントで **STATIC** キーワードが指定されていなければ、すべて動的な配列となります。

静的配列を宣言する **DIM** ステートメントは宣言ステートメントであり、動的配列を宣言する **DIM** ステートメントは実行可能ステートメントです。

STATIC ステートメント, **STATIC** キーワード, **$STATIC** メタコマンドは、いずれも変数や配列に影響を与える目的で使われますが、それぞれ機能は異なります。

種類	機能
SUB, FUNCTION ステートメント中の **STATIC** キーワード	プロシージャ内の変数がデフォルトで静的であることを宣言します。モジュール レベルで共有が宣言されている名前を持つ変数は、モジュール レベルと共有されます。
STATIC ステートメント	指定した変数が静的であることを宣言します。モジュールレベルで共有が宣言されている名前より優先します。
$STATIC メタコマンド	配列がデフォルトで静的であることを宣言します。

参照　**REM** ステートメント，**STATIC** ステートメント

Enabled プロパティ

機能　フォームやコントロールが、ユーザーの操作で発生したイベントを認識するかどうかを決めます。

構文　[*formname.*][*ctlname.*]**Enabled**[= *boolean%*]

boolean%　イベントを認識するかどうかを示す値。 True (-1) は認識することを示し、False (0) はしないことを示します。デフォルトは True (-1) です。

対象　チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，テキスト ボックス，タイマー，垂直スクロール バー

解説　Enabled プロパティを使うと、フォームやコントロールがイベントを認識するかどうかを実行時に切り換えることができます。メッセージを表示するためだけに使うテキスト ボックスなどは、あらかじめ Enabled プロパティを False (0) に設定しておきます。

Enabled プロパティが False (0) に設定されていると、次のようになります。

- コントロールは薄い色で表示され、無効であることをユーザーに示します。
- タブ オーダーから外されます。
- そのフォームまたはコントロールがコンテナとして機能し、子コントロールを持っているときは、子コントロールもイベントを認識しません。また、薄い色で表示されます。

タイマー コントロールの Enabled プロパティを False (0) に設定すると、Interval プロパティの値に従って続けられていたカウントダウンは中断します。True (-1) に戻したときは、カウントダウンは中断時の値ではなく、再び最初から始まります。

参照　Visible プロパティ

END ステートメント

機能

プログラム，プロシージャ，ブロック，ユーザー定義型を終了させるステートメントです。

構文

END[{**DEF**|**FUNCTION**|**IF**|**SELECT**|**SUB**|**TYPE**|*n*%}]

DEF	複数行にわたる **DEF FN** 関数定義を終了させることを示します。
FUNCTION	FUNCTION プロシージャの定義を終了させることを示します。
IF	**IF...THEN...ELSE, IF TYPEOF** ステートメントのブロックを終了させることを示します。
SELECT	SELECT CASE ブロックを終了させることを示します。
SUB	SUB プロシージャを終了させることを示します。
TYPE	ユーザー定義型の定義 (**TYPE** ステートメント) を終了させることを示します。
n%	プログラム終了時に返す値。プログラムを終了させ、この値を返します。省略すると 0 と見なされます。エラーで終了すると -1 が返されます.

解説

キーワードの付かない単独の **END** ステートメントは、プログラムの終了を示します。実行すると、プログラムの実行が終了し、すべてのファイルがクローズされて、オペレーティング システムに 0 を返します。プログラムが終了すると、実行可能プログラムの場合はオペレーティング システムに、プログラミング環境中で実行しているプログラムの場合はプログラミング環境に、それぞれ制御が戻ります。引数 *n*% を指定した場合は、終了コードとして *n*% に指定した値が返されます。

END ステートメントは、次のような位置に必要です。

場所	絶対に必要かどうか
プログラムを終了させたい位置	No
複数行にわたる **DEF FN** 関数定義の最後	Yes
FUNCTION プロシージャ定義の最後	Yes
ブロック **IF...END IF** ステートメントの最後	Yes
ブロック **IF TYPEOF** ステートメントの最後	Yes
SELECT CASE ブロックの最後	Yes
SUB プロシージャの最後	Yes
ユーザー定義型の定義 (TYPE) の最後	Yes

プログラムの終了を示す **END** ステートメントは、プログラム中のどこでも、いくつでも置くことができます。

参照

DEF FN ステートメント，**FUNCTION** ステートメント，**IF...END IF** ステートメント，**SELECT CASE** ステートメント，**STOP** ステートメント，**SUB** ステートメント，**SYSTEM** ステートメント，**TYPE** ステートメント

ENDDOC メソッド

機能

PRINTER オブジェクトにドキュメントの終了を知らせ、プリンタ デバイスを解放します。

構文

PRINTER.ENDDOC

対象

PRINTER オブジェクト

解説

ENDDOC メソッドは、**PRINTER** オブジェクトに、ドキュメントが最後まで出力されたことを伝えます。プリンタ、特にページ プリンタやネットワーク上のプリンタは、この指示を受けて次の動作に移ります。通常は、**PRINT** メソッドによる **PRINTER** メソッドに対する出力が完了した後に使います。

NEWPAGE メソッドの直後に **ENDDOC** メソッドを置くと、余分な空白ページが出力されなくなります。

出力は、**PRINTER** オブジェクトの PrintTarget プロパティに設定されているプリンタまたはファイルに送られます。

参照

NEWPAGE メソッド

ENVIRON ステートメント, ENVIRON$ 関数

機能

ENVIRON$ 関数は、MS-DOS の環境文字列テーブルから、指定した環境変数を検索し、その現在の設定を返します。**ENVIRON** ステートメントは、環境文字列テーブルに対して、環境変数を追加または変更します。

構文

**ENVIRON$** (*env-variable$*)

ENVIRON$ (*n%*)

ENVIRON *string-expression$*

env-variable$	MS-DOS の環境変数名。すべて大文字で指定する必要があります。
n%	環境文字列テーブル中の順番を示す数式。
string-expression$	環境変数名と、その設定を示す文字列。 *string-expression$* 引数には、次のどちらかの書式に従って指定します。環境変数名は、すべて大文字で指定する必要があります。 *環境変数名 = 設定値* *環境変数名 設定値*

解説

次の例は PATH 環境変数を設定します。

```
ENVIRON "PATH=C:¥CLIENTS;C:¥RATES"
```

ENVIRON$** 関数で、*env-variable$* で指定した環境変数が存在しないと、そのテーブルの終わりに追加されます。ENVIRON** ステートメントで、*env-variable$* 引数で指定した環境変数がすでに存在していると、以前の設定は削除され、指定した設定が環境文字列テーブルの最後に追加されます。

ENVIRON ステートメントで、設定値として null 文字列 ("") またはセミコロン (";") を指定すると、その環境変数は削除され、環境文字列テーブル中の残りの環境変数は、順に前に詰められます。

環境変数を追加しようとするとき、環境変数のための領域が不足している場合はエラーになります。不要な環境変数を削除すれば、新たに設定できます。

プログラムの実行が終了すると、追加または変更された環境変数は破棄されます。環境文字列テーブルは、プログラムを実行する前と同じ状態に戻ります。

環境変数名をすべて大文字で指定しないと、(小文字を含む) 指定した名前で環境文字列テーブルに追加されますが、MS-DOS が認識する環境変数としては機能しません。

このステートメントと関数は、**SHELL** ステートメントで起動するプログラムやコマンドなどの "子プロセス" で参照する PATH 環境変数を変更したり、新しい環境変数を使って子プロセスにパラメータを渡すために使うことができます。

ENVIRON$ 関数で、指定した環境変数が設定されていなければ、null 文字列を返します。

ENVIRON$ 関数で数値引数 (*n*) を指定すると、環境文字列テーブルの *n* 番目の環境変数に対応する文字列が返されます。この場合、環境変数名も含めた文字列が返されます。*n* 番目が存在しないときは、null 文字列を返します。引数 *n* には任意の数式を指定することができ、値は整数に丸められます。

EOF 関数

機能

現在位置がファイルの最後なら 0 以外の値を返します。

構文

EOF(*filenumber%*)

filenumber%　　オープンしているファイルのファイル番号。

解説

EOF 関数は、シーケンシャル ファイルから読み出しを行っているとき、読み出し位置がファイルの終わりに達していないかどうか確かめ、"ファイルにもうデータがありません" エラーを避けるために使います。

ランダムアクセス ファイルやバイナリ ファイルで **EOF** 関数を使うとき、最後に実行した **GET** ステートメントがレコード全体を読み込めなかった場合、真 (0 以外の値) が返されます。これは、ファイルの終りを超えてレコードを読み込もうとしたときにも起こります。

EOF 関数を通信デバイスで使うとき、ファイルの終わりの定義はデバイスをオープンしているモード (ASCII, バイナリ) によって異なります。ASCII モードでは、Ctrl+Z を受信するまで偽を返し、それ以降は、デバイスをクローズするまで真を返します。バイナリ モードでは、入力キューが空のとき (LOC(*n*)=0) に真を返し、入力キューが空でないときに偽を返します。

EOF 関数は、Basic のデバイス SCRN:, KYBD:, CONS:, LPT*n*: に対しては使えません。

参照 **LOF** 関数, **LOC** 関数, **OPEN** ステートメント (ファイル I/O)

ERASE ステートメント

機能 静的配列の要素の再初期化と、動的配列に割り当てたメモリの解放を行います。

構文 **ERASE** *arrayname*[, *arrayname*]...

arrayname すでに宣言されている配列の名前。

解説 静的配列に対して **ERASE** ステートメントを実行すると、すべての要素の値は、数値配列では 0、可変長文字列配列は null 文字列になります。また、ユーザー定義型の各要素も、固定長文字列のフィールドも含めて、同様に初期化されます。固定長文字列は、スペースではなく CHR$(0) で初期化されます。

動的配列に対して **ERASE** ステートメントを実行すると、その配列が使っていたメモリを解放します。プログラムが再度この配列を参照するには、**DIM** や **REDIM** ステートメントを使って再び動的配列を宣言しなければなりません。このステートメントで配列を消去せずに、**DIM** ステートメントを使って同じ配列を宣言しようとすると、"配列はすでに宣言されています" エラーが発生します。ただし、**REDIM** ステートメントを使って配列を宣言し直すときは、**ERASE** ステートメントを実行する必要はありません。

参照 **DIM** ステートメント, **REDIM** ステートメント

ERDEV, ERDEV$ 関数

機能

ERDEV 関数は、最後にエラーを起こしたデバイスのエラー コードを返します。**ERDEV$** 関数は、最後にエラーを起こしたデバイスの名前を示す文字列を返します。

構文

ERDEV

ERDEV$

解説

これらの関数はエラーが起こったときだけ意味のある情報を返すので、通常は **ON ERROR** ステートメントで指定したエラー処理ルーチンの中で使います。

ERDEV 関数は、MS-DOS がキャラクタ デバイスまたはブロック デバイスから MS-DOS コールのクリティカル エラーを検出したときに、クリティカル エラー ハンドラ (INT 24H) によって設定されます。また、通信ポートのタイムアウト エラーでもセットされます。

ブロック デバイスのエラーに対しては、**ERDEV** 関数の戻り値の ビット 0～7 (下位バイト) が MS-DOS のエラー コードを示します。ビット 8～15 (上位バイト) は、次のようなデバイス属性情報を含みます。

ビット	値	意味
0 - 7	xxH	MS-DOS エラー コード
8	0	読み出し操作
	1	書き込み操作
9 - 10		影響を受けたディスク領域
	00	MS-DOS
	01	ファイル アロケーション テーブル
	10	ルート ディレクトリ
	11	ファイル領域
11	0	中止 <A> レスポンスが許可されていない
	1	中止 <A> レスポンスが許可されている
12	0	もう1度 <R> レスポンスが許可されていない
	1	もう1度 <R> レスポンスが許可されている
13	0	無視 <I> レスポンスが許可されていない
	1	無視 <I> レスポンスが許可されている
14		未使用
15	0	常に 0

キャラクタ デバイスのエラーに対しては、**ERDEV** 関数の戻り値のビット 0～7 (下位バイト) は MS-DOS のエラー コードを示し、上位バイトは意味を持ちません。

COM のタイムアウト エラーに対しては、**ERDEV** 関数は **OPEN COM** ステートメントのどのオプションがタイムアウトになっているかを示す値を返します。したがって **ERDEV** を下位バイトと上位バイトに分ける必要はありません。

ERDEV の値	OPEN COM オプション タイムアウト
&H80 (128)	CS (CTS タイムアウト)
&H81 (129)	DS (DSR タイムアウト)
&H82 (130)	CD (RLSD タイムアウト)

ERDEV$ 関数は、次の内容を持つ文字列を返します。

- プリンタのようなキャラクタ デバイスのエラーでは、8 バイトのデバイス名。
- 通信ポートがタイム アウトになった場合は、3 バイトのブロック名 (COM)。
- キャラクタ デバイス以外のデバイスでは、2 バイトのブロック名 (A:, B: など)。

次の例は、**ERDEV** の戻り値を下位バイトと上位バイトに分けています。DosErrCode にはブロック デバイスまたはキャラクタ デバイスのエラー コードを保存し、DevAttr にはブロック デバイスの属性情報を保存します:

```
DosErrCode = ERDEV AND &HFF                    'ERDEV の下位バイト
DevAttr    = (ERDEV AND &HFF00) ¥ 256          'ERDEV の上位バイト
```

ERR ステートメント

機能

ERR 関数の戻り値を設定するエラー処理ステートメントです。

構文

ERR = *n*%

n%　エラー コードを示す 1～255 の整数または 0。0 はランタイム エラーが発生しなかったことを意味します。 256 以上の値を指定するとエラーになります。

解説

プログラムを実行中にランタイム エラーが発生したかどうかを調べるためには、**ERR** 関数を使います。実行開始時には、**ERR** 関数が返す値は 0 です。実行中にエラーが発生すると、そのエラー コードが **ERR** 関数にセットされます。**ERR** ステートメントは、この **ERR** 関数の戻す値を、疑似的に設定します。

プロシージャ間でエラー情報をやり取りするには、**ERR** ステートメントを使って **ERR** 関数の戻り値を 0 以外の値に設定します。例えば Visual Basic が使用しないランタイム エラーのコードを、アプリケーション独自のエラー コードとして使用できます。

ERR ステートメント以外にも、次に示す Visual Basic ステートメントが実行されるときに **ERR** が変更されます。

ステートメント	設定される ERR の値
RESUME (任意の形)	0
EXIT SUB, EXIT FUNCTION, EXIT DEF	0 (プロシージャ レベルのエラー処理ルーチン中で実行された場合)
ON ERROR	0
ON LOCAL ERROR	0
ERROR	任意の値 (任意のランタイム エラーを疑似的に再現する場合)

参照　**ERR** 関数，**ERROR** ステートメント

ERR, ERL 関数

機能　**ERR** 関数は、プログラム中で最後に起こったエラー コードを返します。**ERL** 関数はエラーの発生した行の行番号、またはエラーの発生した行よりも前で最も近い行番号を返します。

構文　**ERR**

ERL

解説　**ERR, ERL** 関数は、プログラム中で最後に起こったエラーの状態を知るために使います。**ERR** 関数はエラー コードを返し、**ERL** 関数はエラーの発生した行の行番号を返します。これらの関数は、エラーが起こったときだけ意味のある情報を返すので、通常は **ON ERROR** ステートメントで指定したエラー処理ルーチンの中で使います。これらの関数を使って、発生したエラーの種類を調べ、エラーを回避する処理をするようにしてください。

ERL, ERR は関数なので、これらに値を代入することはできません。ただし **ERR** ステートメントを使うと、関数 **ERR** の返す値を直接指定できます。また **ERROR** ステートメントを使って疑似的にエラーを発生させれば、両方の関数の返す値を間接的に設定することができます。

ERL 関数は、行ラベルではなく、エラーを起こした行の直前の行番号を返します。プログラムで行番号が指定されていない場合、**ERL** 関数はつねに 0 を返します。この場合の **ERL** 関数の機能は、プログラミング環境上で実行している場合と実行可能ファイルにした状態で異なります。実行可能ファイルでは、行番号をエラーが発生した行より前のファイル全体から探します。これに対しプログラミング環境上では、プロシージャでエラーが発生すると、そのプロシージャの中でだけ行番号を探します。プログラミング環境上と実行可能ファイルで同じ動作にするためには、各プロシージャの先頭に行番号を置くようにしてください。

参照　**ERR** ステートメント，**ERROR** ステートメント

ERROR ステートメント

機能

Visual Basic のエラーまたはユーザー定義のエラーを疑似的に発生させるエラー処理ステートメントです。

構文

ERROR *expression%*

expression% エラーコードを示す 1～32767 の整数または 0。

解説

expression% に指定した値が、Visual Basic があらかじめ定義しているエラー コードであれば、**ERROR** ステートメントはそのエラーが発生した場合と同じ状況を作り出します。

ユーザー アプリケーション独自のエラー コードを定義するには、標準で用意されているすべての Basic エラー コードよりも大きな値を指定します。将来の Microsoft Basic エラー コードの増加にそなえて、32767, 32766, 32765, ... といった値を選ぶことをおすすめします。原則として、Visual Basic があらかじめ定義しているエラー コードは 1 から始まる比較的小さな値です。

エラー処理ルーチンが 1 つも有効になっていない状態で **ERROR** ステートメントを実行すると、エラーになり、プログラムの実行が停止します。エラー メッセージが未定義のエラー コードに対しては、エラー メッセージは "未定義エラーです" となります。

参照

ERROR$ 関数

ERROR$ 関数

機能

指定されたエラー コードに対応するエラー メッセージを返します。

構文

ERROR$[(*errorcode%*)]

errorcode% エラー コードを示す 1～32767 の整数。省略すると、最後に発生したランタイム エラーと見なされます。ランタイム エラーが発生していなかった場合は、null 文字列 ("") を返します。

参照

ERR 関数，**ERL** 関数，**ERROR** ステートメント

EVENT ステートメント

機能

イベントのトラッピングを有効または無効にします。フォームやコントロールに関連したイベントには影響を与えません。

構文

EVENT {ON|OFF}

ON 次に **EVENT OFF** が現れるまで、イベントトラップを ON にすることを示します。

OFF 次に **EVENT ON** が現れるまで、イベントトラップを OFF にすることを示します。

解説

EVENT ステートメントは、フォームやコントロールに関連したイベントには影響を与えません。フォームやコントロールを使ってアプリケーションを作成するときは、それらのオブジェクト特有のイベントに対応したイベント プロシージャを記述します。

イベント トラップを行うステートメントを含むプログラムを BC コマンドラインからコンパイルするとき、/W または /V オプションを付けずにコンパイルすると、"ON event を含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

イベントトラップの有無は、コンパイルしたコードの実行スピードとサイズに影響します。プログラム中に **EVENT OFF** が現れると、次に **EVENT ON** が現れるまでの間、イベント チェック用のコードが生成されなくなります。これは、/V や /W を指定せずに、プログラムをコマンドラインからコンパイルしたときと同じ状態です。イベントトラップが不必要な場所でトラップのためのコードが生成されないようにすると、プログラムをより小さく高速にすることができます。**EVENT OFF** を実行しても、トラップ用のコードが生成されないだけで、イベントの検出はつねに行われています。したがって次に **EVENT ON** を実行すると、そのイベントはトラップされます。

EVENT ステートメントの実行は、プログラム全体にグローバルに影響します。

EXIT ステートメント

機能

DEF FN 関数，**DO...LOOP** および **FOR...NEXT** ループ，FUNCTION および SUB プロシージャを抜けるフロー制御ステートメントです。

構文

EXIT {DEF|DO|FOR|FUNCTION|SUB}

DEF **DEF FN** 関数の実行を抜け、**DEF FN** 関数を呼び出した位置に制御を移すことを示します。**DEF FN** 関数の中でだけ使うことができます。

DO	**DO...LOOP** ループを抜け、**LOOP** ステートメントの次のステートメントに制御を移すことを示します。**DO...LOOP** ループの中でだけ使うことができます。**DO...LOOP** ループがネストされている場合は、もっとも内側のループだけを抜けます。
FOR	**FOR...NEXT** ループを抜け、**NEXT** ステートメントの次のステートメントに制御を移すことを示します。**FOR...NEXT** ループの中でだけ使うことができます。**FOR...NEXT** ループがネストされている場合は、もっとも内側のループだけを抜けます。
FUNCTION	FUNCTION プロシージャを抜け、FUNCTION プロシージャを呼び出した位置に制御を移すことを示します。FUNCTION プロシージャの中だけで使うことができます。
SUB	SUB プロシージャを抜け、SUB プロシージャを呼び出したステートメントの直後のステートメントに制御を移すことを示します。SUB プロシージャの中だけで使うことができます。

解説

EXIT ステートメントはループやプロシージャを抜け出す方法の 1 つで、これらの終わりを定義するものではありません。

参照

DEF FN ステートメント，**DO...LOOP** ステートメント，**FOR...NEXT** ステートメント，**FUNCTION** ステートメント，**SUB** ステートメント

EXP, LOG 関数

機能

EXP 関数は、指数関数 (e を底とする数式のべき乗) を計算する数値演算関数です。**LOG** 関数は、自然対数を計算する数値演算関数です。

構文

EXP(*numeric-expression*)

LOG(*numeric-expression*)

numeric-expression　計算対象の数値。

解説

numeric-expression は、いずれの数値データ型でもかまいません。*numeric-expression* が整数または単精度の値であると単精度で計算します。その他の数値データ型であると、倍精度で計算します。

EXP 関数の *numeric-expression* は、単精度のときは 88.02969 以下、倍精度のときは 709.782712893 以下の値でなければなりません。この範囲外の値を指定すると、実行時に "オーバーフローしました" エラーが発生します。

LOG 関数の *numeric-expression* は、正の値でなければなりません。

自然対数は、e を底とする対数です。定数 e は、約 2.718282 です。

任意の数 x の、n を底とする対数は、x の自然対数を n の自然対数で割ることで計算できます。例えば、次の FUNCTION プロシージャは 10 を底とする対数を計算します。

```
FUNCTION LOG10(X) STATIC
    LOG10 = LOG(X)/LOG(10)
END FUNCTION
```

FIELD ステートメント

機能

ランダムアクセス ファイルのバッファに、変数のための領域を割り当てます。

構文

FIELD [#]*filenumber%*, *fieldwidth%* **AS** *stringvariable$*[, *fieldwidth%* **AS** *stringvariable$*] ...

filenumber%	オープンしているファイルのファイル番号。
fieldwidth%	フィールドのバイト数。
stringvariable%	フィールドのデータを格納する文字列変数。

解説

FIELD ステートメントは、以前のバージョンとの互換性を保つために残されています。ユーザー定義型変数と **OPEN** ステートメントを使うと、ランダムアクセス ファイルをより容易に実現できます。

FIELD ステートメントで割り当てるバイト数の合計が、ファイルをオープンするときに指定したレコード長を超えないようにしてください。レコード長を超えると、"FIELD 長を超えました" エラーになります。デフォルトのレコード長は、128 バイトです。

同じファイルに対して、いくつ **FIELD** ステートメントを実行してもかまいません。複数の **FIELD** ステートメントで定義されたフィールドは、同時に利用することができます。

ファイルをクローズすると、そのファイルに対して定義したすべてのフィールドが削除されます。このとき、フィールドに割り当てられていた文字列は、すべて null 文字列になります。

フィールドとして定義した変数の値を残しておきたいときに、**INPUT** ステートメントや代入ステートメントで、その変数名を使わないでください。フィールドとして定義された変数のポインタは、ランダムアクセス ファイルのバッファ中の任意の位置を指します。このあとに、**INPUT** ステートメントや代入ステートメントで同じ変数名を指定すると、変数のポインタはランダムアクセス ファイルのバッファではなく、文字列領域を指すようになります。

参照

GET ステートメント (ファイル I/O), **PUT** ステートメント (ファイル I/O), **OPEN** ステートメント (ファイル I/O)

FILEATTR 関数

機能

オープンしているファイルのファイル ハンドルまたはファイル モードを返します。

構文

FILEATTR(*filenumber%*, *attribute%*)

filenumber% オープンしているファイルのファイル番号。

attribute% 取得したいファイル情報の種類を示す、1 または 2 の値。1 を指定するとファイルのモードを表すコードを返します。2 を指定するとファイル ハンドルを返します。

解説

attribute% に 1 を指定すると、指定したファイルのモードを示す、次のような値を返します。

戻り値	モード
1	入力モード (**INPUT**)
2	出力モード (**OUTPUT**)
4	ランダムアクセスモード (**RANDOM**)
8	追加モード (**APPEND**)
32	バイナリモード (**BINARY**)

attribute% に 2 を指定すると、指定したファイルのファイル ハンドルを返します。この値は、ファイル操作を行う MS-DOS のシステム ファンクションを直接呼び出すときに使います。

参照

OPEN ステートメント (ファイル I/O)

FileName プロパティ

機能

ファイル リスト ボックスでそのとき選択されているファイル名の設定と取得を行います。Path, Pattern の各プロパティも同時に変更可能です。デザイン時には無効です。

構文

[*formname*.]*ctlname*.**FileName**[= *filename$*]

ctlname ファイル リスト ボックスの名前。

filename$ ファイル名を示す文字列式。ドライブ名，ディレクトリ パスを含めることができ、ワイルド カードを含むパターンを指定することもできます。デフォルトは null 文字列 ("") です。

対象

ファイル リスト ボックス

解説

実行時には、FileName プロパティはファイル リスト ボックスで現在反転表示されているファイルの名前を返します。反転表示されているものがなければ、null 文字列 ("") を返します。ファイル リスト ボックスの名前が File1 であれば、File1.FileName が返す値は、機能的には次に示す式の値と同じです。

```
File1.List(File1.ListIndex)
```

Path プロパティを使うと、そのとき選択されているディレクトリ パスだけを調べることができます。また、そのとき設定されている (ワイルド カードを含む) パターンを調べるには、Pattern プロパティを使います。

コード中から FileName プロパティの設定を変更する場合、*filename$* 引数で指定した文字列にディレクトリ パスや (ワイルド カードを含む) パターンが含まれている場合は、それぞれ Path, Pattern の各プロパティも同時に変更されます。このとき、ディレクトリ パスが変更されれば PathChange イベント，パターンが変更されれば PatternChange イベントが発生します。

コード中から FileName プロパティに対して、そのとき存在しているファイルの名前をパターンを使わずに直接設定したときは、そのファイルが反転表示されます。このとき、DblClick イベントが発生します。

参照

Path プロパティ，Pattern プロパティ

FILES ステートメント

CHDIR ステートメントを参照してください。

FIX, INT 関数

機能

FIX 関数は、引数に指定した数式の小数部を切り捨て、整数部だけを返す数値演算関数です。**INT** 関数は、引数に指定した数式の値以下で、最大の整数を返す数値演算関数です。

構文

FIX(*numeric-expression*)

INT(*numeric-expression*)

numeric-expression　　任意の数式。

解説

FIX(x) は SGN(x)*INT(ABS(x)) と同じ意味です。したがって、x が正である場合の **FIX** 関数は、**INT** 関数と同じ機能です。x が負の場合は、**INT** 関数が x より小さい最大の負の整数を返すのに対し、**FIX** 関数は x より大きい最小の負の整数を返します。

INT 関数は、指定した引数の小数部分を削除します。数値を四捨五入するには、この関数を含む、次のような式を使ってください。

INT(*numeric-expression* + 0.5)

参照 **CINT** 関数

FOR...NEXT ステートメント

機能 指定した回数だけ、一連のステートメントの実行を繰り返すフロー制御ステートメントです。

構文

FOR *counter* = *start* **TO** *end*[**STEP** *increment*]

[*statementblock*]

[**EXIT FOR**]

[*statementblock*]

NEXT[*counter*[, *counter*]...]

counter	ループカウンタに使う数値変数。配列変数やユーザー定義型の要素を指定することはできません。
start	ループカウンタの初期値。
end	ループカウンタの最終値。
STEP *increment*	*increment* を指定することを示します。**STEP** *increment* を省略すると、ループを繰り返すごとにループカウンタには 1 が加算されます。
statementblock	ループ内で実行される、任意の行数のステートメント。
EXIT FOR	ループを終了し、**NEXT** キーワードの次のステートメントに移ることを示します。

解説 **NEXT** の引数を省略すると、1 番最後に実行された **FOR** に対応する **NEXT** だと解釈されます。**NEXT** が、対応する **FOR** の前に実行されると、"NEXT がありません" エラーが発生します。

EXIT FOR は、**FOR...NEXT** ループの中でだけ使うことができます。

ネストした **FOR...NEXT** ループの中で **EXIT FOR** を実行すると、それを囲んでいる 1 番内側のループから抜け出します。

FOR...NEXT ループは、引数 *start* と *end* に対して、正しい *increment* が指定されているときにだけ実行されます。*end* が *start* より大きいとき、*increment* は正の数でなければなりません。*end* が *start* より小さいときは、*increment* は負の数でなければなりません。この関係は、実行時に (*end* - *start*) の符号と *increment* の符号を比較してチェックされます。両者の符号が同じであると、プログラムの実行は **FOR...NEXT** ループに入り、符号が違っていると **FOR...NEXT** ループはスキップされます。

FOR...NEXT ループの中では、**FOR** 以降の行が、**NEXT** に達するまで実行されます。実行が **NEXT** に達すると、*counter* に *increment* の変化が加えられたあと、*counter* と終了値 *end* の値が比較されます。

increment の値が正または *increment* が省略されたときは、*counter* の値が *end* 以下のとき、ループが繰り返し実行されます。*counter* の値が*end* より大きくなると、ループを抜け出して **NEXT** の次のステートメントに制御が移ります。*increment* の値が負の場合は、この逆の動作になります。

start と *end* が同じ値であると、*increment* の値に関係なく、ループは一度だけ実行されます。*increment* の値が 0 の場合、ループは無限に繰り返されます。

counter の値をループ内で意図的に変更することは避けてください。ループ カウンタの値を変更すると、プログラムの理解やデバッグが困難になります。

FOR...NEXT ループはネストできます。つまり、**FOR...NEXT** ループの内部に別の **FOR...NEXT** ループを入れることができます。ループをネストするときは、それぞれのループカウンタに別の変数を指定してください。内側のループの **NEXT** は、外側のループの **NEXT** よりも前になければなりません。

次のように、1 つの **NEXT** に、複数のループカウンタを指定することもできます。指定する順番に注意してください。

```
NEXT K, J, I
```

参照　**DO...LOOP** ステートメント，**WHILE...WEND** ステートメント

ForeColor プロパティ

BackColor プロパティを参照してください。

$FORM メタコマンド

機能

別のフォームのコントロールおよびプロパティへアクセスできるようにするメタコマンドです。すべての実行可能ステートメントより前に置かなければなりません。

構文

{'|REM} **$FORM** *formname*

formname フォームの名前。

すべてのメタコマンドは行頭にシングル クォーテーション (') または **REM** を付け、コメント行として記述します。

解説

通常、あるフォーム モジュールからは、そのフォーム モジュールに含まれるフォームやそれ上のコントロール、およびそれらのプロパティにだけアクセスできます。**$FORM** メタコマンドを使うと、別のフォーム モジュール中のフォームやコントロール，プロパティにアクセスすることができます。対象となるフォームの名前を *formname* 引数で指定してください。1 つの **$FORM** メタコマンドで指定できるのは 1 つのフォームだけです。複数のフォーム モジュールに対してアクセスしたいときは、複数の **$FORM** メタコマンドを記述する必要があります。

複数のフォームを使うプロジェクトをプログラミング環境で開発する場合、プログラムを実行すると、フォーム モジュール間での参照関係が解析され、フォーム モジュールの先頭に自動的に **$FORM** メタコマンドが挿入されます。

複数のフォームを使うプロジェクトでは、複数の **$FORM** メタコマンドを 1 つのインクルード ファイルにまとめておき、各モジュールで **$INCLUDE** メタコマンドを使って、そのファイルをインクルードするようにすると便利です。

参照

$INCLUDE メタコマンド，**REM** ステートメント，**SHOW** メソッド

FORMAT$ 関数

機能

指定した書式に従って、数値を文字列に変換します。

構文

**FORMAT$**(*expression*[, *fmt$*])

expression 変換元の数値を示す数式。

fmt$ 日付/時刻や数式の書式を指定する文字を含む文字列式。

解説

*fmt$* を省略したときや null 文字列を指定したとき、**FORMAT$** 関数は **STR$** 関数と同じ機能になり、正の数値を文字列に変換します。ただし、**STR$** 関数で変換した場合には数値を示す文字列の前に符号用のスペースが加えられるのに対し、**FORMAT$** 関数で変換した結果の文字列の前には加えられません。表示書式が問題でない場合は、数値を文字列に変換するためには **STR$** 関数を使ってください。

書式指定文字列 (*fmt$*) には、セミコロン (;) で区切ってセクションを 3 つまで含めることができます。最初のセクションは正の数値、次のセクションは負の数値、最後のセクションは 0 の書式をそれぞれ指定します。

セクション数	結果
1 セクションのみ	書式はすべての数値に適用されます。
2 セクション	最初のセクションは正の数値と 0 に適用され、次のセクションの書式は負の数値に適用されます。
3 セクション	最初のセクションは正の数値、次のセクションは負の数値、最後のセクションは 0 に適用されます。

セミコロンを続けて指定すると、省略されたセクションに対応するタイプの数値は表示されません。

数値データの書式指定

文字	規則
null 文字列	特に書式を指定せずに表示します。
0	1 つを 1 桁として、桁位置と桁数を指定します。それぞれの桁位置はつねに埋められます。もし指定された桁数 (0 の数) よりも出力される数字の桁数が少ない場合は、小数点以上と以下のどちらであっても、 0 で埋められます。小数点以下の数値の桁数が多いときは、書式で指定した桁数に合わせて丸められます。小数点以上の数値の桁数は、書式で指定した桁数より多くても必ず表示されます。
#	0 と同じ機能ですが、指定された桁数 (0 の数) よりも出力される数字の桁数が少ない場合でも、0 で埋められません。
.	小数点の位置を指定します。小数点の左側に桁位置 (0 または #) が指定されていなければ、1 未満の数値は小数点で始まります。小数点の前に 0 を付けるには、1 の位の桁位置に # ではなく 0 を使ってください。
%	数値を 100 倍して、パーセント記号 % 付きで表示することを指定します。
,	小数点以上 3 桁ごとに置き、桁区切りのカンマを表示します。小数点より左側にカンマを 2 つ続けて置くか、小数点のすぐ左にカンマを置くと (数値の桁位置を指定したかどうかにかかわらず)、カンマの間またはカンマと小数点の間の桁は省略され、数値は丸められます。これにより、桁数の多い数値の表示を簡略化できます。例えば "##0,,." と指定すると、100,000,000 は 100 として表示されます。このとき 1,000,000 (1 million) 未満の数値は 0 と表示されます。

E- E+ e- e+	数値を指数表記で表示することを指定します。 0 または # の間に E-, E+, e-, e+ のいずれかを置くと、数値は指数表記で表示され、仮数部と指数部の間に E または e が表示されます。E または e より左側の 0 や # の数は仮数部の桁数を示し、右側の 0 や # の数は指数部の桁数を示します。負の指数にマイナス記号 (-) を付けるには、E- または e- を使います。正の指数にはプラス，負の指数にマイナスを付けるには、 E+ または e+ を使います。
¥	すぐ後に続く 1 文字をそのまま表示します。書式指定の中で特別な意味を持っている # や E などの文字をそのまま表示するには、その文字の前に円記号を付けます。文字をダブル クォーテーションで囲んでも、文字がそのまま表示されます。 特別な意味を持つ文字には、日付 /時刻表示の書式指定に使う文字 (y, m, d, h, s, a, p など) や数値の書式指定に使う文字 (#, 0, %, E, e, カンマ, ピリオドなど) があります。
"abc"	ダブル クォーテーションに囲まれた文字をそのまま表示します。ダブル クォーテーションは、 CHR$(34) と指定できます (34 はダブル クォーテーションの ASCII コードです)。

日付/時刻を表すシリアル値は、浮動小数点数として記憶されています。このため、日付/時刻を表すシリアル値に対しては、日付/時刻表示用の書式と数値用の書式のどちらでも指定することができます。

日付および時刻データの書式指定

文字	意味
:	時刻表示で、時，分，秒の区切りを示します。
/	日付表示で、年，月，日の区切りを示します。
yy	西暦を 2 桁で表示します (00-99)。
yyyy	西暦を 4 桁で表示します (1900-2040)。
m	1 桁の場合も先頭に 0 を付けずに月を表示します (1-12)。 h または hh の直後に指定したときは、月でなく分を表示します。
mm	1 桁の場合は先頭に 0 を付けて月を表示します (01-12)。 h または hh の直後に指定したときは、月でなく分を表示します。
mmm	月を略称で表示します (Jan～Dec)。
mmmm	月をフル スペルで表示します (January～December)。
d	1 桁の場合も先頭に 0 を付けずに日を表示します (1-31)。
dd	1 桁の場合は先頭に 0 を付けて日を表示します (01-31)。
ddd	曜日を略称で表示します (Sun～Sat)。
dddd	曜日をフル スペルで表示します (Sunday～Saturday)。
ddddd	日付のシリアル値を年，月，日を含む完全な日付として表示します。
h	1 桁の場合も先頭に 0 を付けずに時を表示します (0-23)。
hh	1 桁の場合は先頭に 0 を付けて時を表示します (00-23)。
m	1 桁の場合も先頭に 0 を付けずに分を表示します (0-59)。 h または hh の直後に指定したとき以外は、分でなく月を表示します。

mm	1 桁の場合は先頭に 0 を付けて分を表示します (00-59)。 h または hh の直後に指定したとき以外は、分でなく月を表示します。
s	1 桁の場合も先頭に 0 を付けずに秒を表示します (0-59)。
ss	1 桁の場合は先頭に 0 を付けて秒を表示します (00-59)。
ttttt	時刻のシリアル値を、時，分，秒を含む完全な時刻として表示します。
AM/PM am/pm	(デフォルト) 12 時間制で表示します。正午以前の時刻には AM または am、正午以後 11:59 までの時刻には PM または pm を表示します。
A/P a/p	12 時間制の時計を使います。正午以前の時刻には A または a, 正午以後 11:59 までの時刻には P または p を表示します。

FormName プロパティ

機能

フォームを識別するための名前を決めます。実行時には取得および設定できません。

対象

フォーム

解説

フォームを生成した直後のデフォルトの名前 (FormName プロパティのデフォルト値) は、"Form" に順に番号を付けたもの (Form1, Form2, Form3 など) になります。番号は 1 から順に増えていきます。これにより、すべてのフォームにはデフォルトでそれぞれ異なる名前が付けられます。

フォームの名前を変更する場合は、他の名前と重複しないようにする必要があります。フォーム名は、アルファベットで始まり、アルファベット，数字，アンダースコア (_) だけで構成された、40 文字以内の文字列でなければなりません。また、"フォーム名" + "_" + "最も長いイベント名またはプロパティ名" が 40 文字以内になるように注意してください。

Visual Basic の予約語は、FormName プロパティには設定できません。

FormName プロパティと Caption プロパティを混同しないように注意してください。デフォルトでは 2 つのプロパティに同じ文字列が設定されますが、識別子として使われるのは FormName プロパティです。

フォーム モジュール中では、フォーム自身に対してメソッドを使ったり、プロパティを参照するとき、フォーム名は省略できます。例えば、フォーム名、つまり FormName プロパティの値が Form1 のフォーム モジュールでは、次の 2 つの行は同じ意味です。

```
Form1.PRINT "Hello"
PRINT "Hello"
```

フォーム モジュール中から別のフォームを参照するときや、コード モジュール中からフォームを参照するときは、対象のフォームを示す必要があるため、フォーム名は省略できません。

プログラム中で FormName プロパティ自身を参照することはできません。例えば次の行はエラーになります。

```
FOO$ = Form1.FormName
```

フォームを引数としてプロシージャに渡すときなど、どのフォームであるかを判別する必要があるときは、FormName プロパティではなく、Tag プロパティを使ってください。

参照

CtlName プロパティ，Tag プロパティ

FormType プロパティ

機能

フォームが MDI のコンテナ フォームであるかどうかを決めます。デザイン時のみ有効です。

対象

フォーム

解説

FormType プロパティはデザイン時にのみ設定可能で、実行時には設定，参照ともにできません。0 がノーマル，1 が MDI のコンテナ フォームを示し、デフォルトは 0 です。

FormType プロパティが 1 に設定されているフォームは MDI のコンテナ フォームとして機能し、アプリケーションの他のフォームは子フォームとして扱われます。

FormType プロパティを 1 に設定できるのは、1 つのアプリケーション (プロジェクト) 中で 1 つだけです。複数のフォームの FormType プロパティを MDI (コンテナ フォーム) に設定しようとすると、エラーが発生します。

コンテナ フォームとして機能しているフォーム (FormType プロパティが True (-1) であるフォーム) は、通常のフォームと次の点が異なります。

- フォームは常に最大化されています。移動，サイズ変更，最小化，復元の各操作はできません。実行時に **MOVE** メソッドは無効です。
- 境界のスタイル (BorderStyle プロパティ) を 0 以外に設定し、コントロール メニュー ボックスが表示されている場合、コンテナ フォームをクローズすると、他のすべてのフォームのクローズされて、アプリケーションが終了します。
- MinButton, MaxButton, WindowState, Height, Width, Top, Left の各プロパティは、デザイン時，実行時ともに読み込み専用です。

コンテナ フォームとして機能しているフォームが持つことのできるコントロールは、メニュー コントロールだけです。子フォームに対してメニューを設定することはできますが、実行するとコンテナ フォーム上に表示されます。複数の子フォームがあるときは、そのときアクティブな子フォームのメニューだけが表示されます。

プロジェクト中の他のすべてのフォームは、コンテナ フォーム (親フォーム) の子フォームになります。このとき、次のルールが適用されます。

- コンテナ フォームをロードしても、その子フォームは自動的にはロードされません。逆に、子フォームをロードするとコンテナ フォームは自動的にロードされます。コンテナ フォームをアンロードすると、子フォームもすべてアンロードされます。
- コンテナ フォームが表示されていなければ、Visible プロパティの設定にかかわらず、すべての子フォームは表示されません。
- コンテナ フォームを非表示にすると、Visible プロパティの設定にかかわらず、すべての子フォームも非表示になります。コンテナ フォームを再び表示したときは、Visible プロパティが True (-1) に設定されている子フォームだけが表示されます。
- 子フォームの Enabled プロパティが True (-1) であっても、コンテナ フォームの Enabled プロパティが False (0) なら、その子フォームはイベントを認識しません。また、コンテナ フォームの Enabled プロパティを True (-1) に設定したときは、Enabled プロパティが True (-1) の子フォームだけがイベントを認識します。

FRE 関数

機能

使用可能なメモリ容量をバイト単位で返すメモリ管理関数です。

構文

FRE(*numeric-expression%*|*string-expression$*)

numeric-expression% -1, -2, -3, -4 のいずれか。この値は、スタック領域など、獲得したい情報の種類を示します。

string-expression$ 任意の文字列定数か文字列変数を指定します。 **FRE** 関数は、指定した引数に従って、次のような値を返します。

FRE(a$):	その文字列を含むセグメント (バイト単位)
FRE("リテラル文字列"):	テンポラリ文字列領域 (バイト単位)
FRE(0):	"引数が有効範囲にありません" エラーが発生
FRE(-1):	データまたはコードの far メモリ (バイト単位)
FRE(-2):	スタック領域 (バイト単位)
FRE(-3):	EMS メモリ (K バイト単位)
FRE(-4):	カレント フォームのデータ セグメント (バイト単位)
FRE(任意の他の数値):	"引数が有効範囲にありません" エラーが発生

解説

引数として文字列式を指定すると、文字列領域の未使用部分が整理されて 1 つにまとめられ、文字列領域の使用可能な量が返されます。

FRE(-2) の返す値は、プログラムの実行中でしか意味を持ちません。イミディエイトウィンドウから関数を呼び出したときや、プログラムをトレースしているとき、または変数の値をモニタしているときは、FRE(-2) が返す値は正確ではありません。スタックの未使用領域は、DGROUP を再利用するためにどれだけスタックが必要かを示します。

EMS メモリがない場合に FRE(-3) を実行すると、"この機能はサポートされていません" エラーが発生します。

/E:n オプションを指定せずに Visual Basic (VBDOS.EXE) を起動すると、使用可能な EMS メモリすべてを利用します。

Visual Basic (VBDOS.EXE) 起動時に /E:n オプションを指定すると、EMS の使用量を制限できます。例えばシステムに 5MB の EMS メモリがあり、そのうち 4MB を使いたいときは、次のように Viual Basic を起動します。

```
VBDOS /E:4096
```

このように起動して 4MB の EMS メモリを獲得し、プログラム中で 2MB の EMS メモリを使用すると、FRE(-3) の戻り値は 2048 になります。

以前のバージョンの Basic の、EMS オーバーレイ機能を利用した実行可能ファイルを実行する場合は、FRE(-3) の返す値は使用可能な EMS メモリの総量に基づいて決定されます (オーバーレイを使用する場合は、プログラムの使う EMS メモリの利用を制限できません)。

FRE 関数を使って、フォームやフォーム上のコントロールが占めるメモリのおよその量を知ることができます。

```
SUB Form_Click ()
    Loaded& = FRE(-1)
    UNLOAD Form1
    Unloaded& = FRE(-1)
    SHOW
    PRINT STR$(Unloaded& - Loaded&)
END SUB
```

これと同じようなイベント プロシージャを作成し、コントロールをフォームに置く前と後を調べることによって、そのコントロールが占めるメモリの量を知ることができます。

A = コントロールを含まないデフォルトのフォームのサイズ (単位: バイト)
B = 1 個のコントロールを含むフォームのサイズ (単位: バイト)
B - A = コントロールの占めるメモリ量 (単位: バイト)

FREEFILE 関数

機能 次に使用可能なファイル番号を返します。

構文 **FREEFILE**

解説 この関数を使えば、SUB および FUNCTION プロシージャで、すでに使われているファイル番号を重複して使うのを防ぐことができます。

参照 **OPEN** ステートメント (ファイル I/O)

FUNCTION ステートメント

機能 FUNCTION プロシージャの名前，引数とそのデータ型，戻り値のデータ型を宣言し、FUNCTION プロシージャの始まりを示します。

構文

[**STATIC**]**FUNCTION** *name* ([*parameterlist*])[**AS** *type*]

[*statementblock*]

name = *expression*

[**EXIT FUNCTION**]

[*statementblock*]

name = *expression*

END FUNCTION

STATIC プロシージャ中のローカル変数の値が、次に呼び出されるまで保存されることを示します。省略すると、ローカル変数は、プロシージャが呼び出されるたびに値が代入され、プロシージャの実行が終了すると値が失われます。プロシージャの外 (モジュールレベル) で宣言された変数には影響しません。

name 定義する関数の名前 (FUNCTION プロシージャ名)。プログラム中ですでに使われている名前は指定できません。プロシージャ名は、Visual Basic の変数の名前付け規約に従います。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。

parameterlist 呼び出し側との受け渡しに使う変数とそのデータ型を、カンマで区切って指定します。

type 定義する関数が返す値のデータ型。 **INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING** のいずれかを指定します。

statementblock その FUNCTION プロシージャの中で実行される、任意の行数のステートメント。

expression 定義する関数の戻り値を計算する式。ユーザー定義関数は、値を関数名に代入することで値を返します。

EXIT FUNCTION FUNCTION プロシージャを終了し、呼び出し側に値を返します 。1 つの FUNCTION プロシージャ中に複数の **EXIT FUNCTION** を記述することもできます。

parameterlist 引数には、次の構文に従って引数とそのデータ型を指定します。

[**BYVAL**]*variable*[()][**AS** *type*][, [**BYVAL**]*variable*[()][**AS** *type*]]...

BYVAL 直後に指定されている変数に対して、デフォルトの引き渡し方法である参照による引き渡しではなく、値による引き渡しで渡されることを示します。**BYVAL** は、数値の基本データ 型 (**INTEGER, LONG, SINGLE, DOUBLE, CURRENCY**) と 可 変 長 文 字 列 (**STRING**) に対してだけ指定できます。

variable 渡された引数を受け取る変数の名前。配列変数の場合、名前の後ろに空のカッコ () を付けます。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。ユーザー定義型に含まれるものでなければ、固定長文字列変数を指定することはできません。

type 変数のデータ型。変数それぞれに対して指定します。 **INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING**, ユーザー定義型, **FORM, CONTROL** のいずれかが指定できます。 **FORM** は引数としてフォームが渡されることを示し、 **CONTROL** はいずれかのコントロールが渡されることを示します。

解説

FUNCTION と **END FUNCTION** は、FUNCTION プロシージャの始まりと終わりに記述します。また、プロシージャ内の任意の場所に、FUNCTION プロシージャの実行を終了させる **EXIT FUNCTION** ステートメントを置くことができます。

FUNCTION プロシージャは、SUB プロシージャと同様に、独立したプロシージャです。引数を受け取り、一連のステートメントを実行し、引数の値を変更します。ただし SUB プロシージャとは異なり、FUNCTION プロシージャは (**SQR, COS, CHR$** といった) Visual Basic 固有の関数のように数式や文字列式の中で使うことができます。

プロシージャ中で使われる変数は、**FUNCTION** ステートメントで引数として指定された変数を除き、原則としてローカル変数として扱われます。ただし、**SHARED** ステートメントで共有変数として宣言された変数や、**DIM** または **COMMON** ステートメントで **SHARED** キーワードを指定して宣言された変数は、ローカル変数ではありません。

FUNCTION プロシージャがフォーム モジュールに含まれているときは、定義された関数は、そのフォーム モジュール中のどこからでも呼び出すことができます。(FUNCTION プロシージャ名は、そのフォーム モジュールに対してはグローバルです)。ただし、他のモジュールから呼び出すことはできません。コード モジュールに含まれているときは、プログラム全体のどこからでも呼び出すことができます (プログラム全体に対してグローバルです)。したがって、同じ名前を変数やほかのプロシージャの名前にすることはできません。

FUNCTION プロシージャが返す値のデータ型は、次の条件によって決まります。

- 関数名 (FUNCTION プロシージャの名前) の最後に型宣言文字 (%, &, !, #, @, $) が付けられていれば、それが示すデータ型。この場合、**FUNCTION** ステートメントの最後に **AS** キーワードで指定することはできません。
- **FUNCTION** ステートメントの最後に **AS** キーワードでデータ型が指定されていれば、そのデータ型。この場合、関数名に型宣言文字を付けることはできません。
- どちらも指定されていない場合、**DEF***type* ステートメントで先頭の文字が設定されてれば、そのデータ型。**DEF***type* ステートメントが使われていなければ、単精度浮動小数点 (**SINGLE**) 型。

FUNCTION プロシージャ内で関数名に対して値を代入しないと、戻り値として、数値関数では 0、文字列関数は null 文字列 ("") を返します。

変数に対して **BYVAL** キーワードが指定されていなければ、実引数に変数が指定されると、参照によって引き渡されます。プロシージャ内でその変数の値を変更すると、呼び出し側の値も変更されます。実引数に式が指定されたときは、常に値で引き渡され、呼び出し側の変数には影響しません。**BYVAL** キーワードが指定されていなくても、プロシージャを呼び出すときに変数をカッコで囲んで指定すると、式と見なされて値で渡されます。

STATIC キーワードを使うと、プログラムの実行速度がわずかに上がります。ただし、メモリの使用効率が落ちる場合があります。

FUNCTION プロシージャは再帰呼び出しできます。つまり、自分自身を呼び出せます。再帰的な FUNCTION プロシージャでは、**STATIC** キーワードは指定しません。

Visual Basic では、文字列を FUNCTION プロシージャに渡す前に、単純固定長文字列引数を可変長文字列引数に変換するため、FUNCTION プロシージャを呼び出すときの引数 (実引数) として固定長文字列の配列を使うことはできません。

Basic は、演算の効率を高めるため、数式を並べ替えることがあります。このため、変数の値を変えてしまうような FUNCTION プロシージャは、数式の中で使わないでください。

入出力を行うステートメントから呼び出される FUNCTION プロシージャでは、入出力ステートメントを使わないでください。使用すると結果は予測できません。

GOSUB, **GOTO**, **RETURN** によって、FUNCTION プロシージャに入ったり、FUNCTION プロシージャから抜けたりすることはできません。

参照　**CALL** ステートメント (Basic プロシージャ), **DECLARE** ステートメント, **SUB** ステートメント

GET, PUT ステートメント (ファイル I/O)

機能

GET ステートメントは、ディスク ファイルの内容を、ランダムアクセス ファイルのバッファや変数に読み込みます。**PUT** ステートメントは、変数やランダムアクセス ファイルのバッファの内容を、ファイルに書き込みます。

構文

GET [#]*filenumber%*[, [*recordnumber&*][, *variable*]]

PUT [#]*filenumber%*[, [*recordnumber&*][, *variable*]]

filenumber%	オープンしているファイルのファイル番号。
recordnumber&	読み出すまたは書き込むレコードの番号 (ランダムアクセス ファイル) またはファイル中のバイト位置 (バイナリ ファイル)。ファイル中の最初のレコードやバイト位置は、1 になります。
variable	ファイルから読み出した値を格納する変数 (**GET**) またはファイルに書き込む値を持つ変数 (**PUT**)。通常、ユーザー定義型です。

解説

recordnumber& を省略すると、次のレコードまたはバイト (直前に実行した **GET, PUT** ステートメントで読み書きしたレコードまたはバイトの次のもの、あるいは直前に実行した **SEEK** ステートメントが指すレコードまたはバイト) がバッファに読み込みまたは書き込まれます。レコード番号の最大値は 2,147,483,647 (2^{31} -1) です。

1 つのレコードは、32,767 バイト以下でなければなりません。

variable を指定すると、次の操作は不要になります。また、そのファイルに対して **FIELD** ステートメントは実行できません。

- レコード フィールドを数値に変換するための **CVI, CVL, CVS, CVD, CVC**
- 書き込み前に数値フィールドを変換するための **MKI$, MKL$, MKS$, MKD$, MKC$**

FIELD ステートメント

ランダムアクセス ファイルでは、レコード長を超えなければ、どんな長さの変数でも使うことができます。通常は、データ レコードのフィールドに合せて定義されたユーザー定義型の変数を使います。

バイナリ ファイルでは、どんな変数でも使うことができます。**GET** ステートメントは、どんなに長い変数でも読み込むことができます。**PUT** ステートメントは、その変数のすべての内容をファイルに書き込みます。

可変長文字列変数を使うとき、これらのステートメントは、文字列変数の持つ値の文字数と同じバイト数を読み込みまたは書き込みます。例えば、次のステートメントは、ファイル番号 1 のファイルから 10 バイトを読み込みます。

```
VarStrings$=STRING$ (10, " ")
GET #1,,VarString$
```

次のステートメントでは、ファイル番号 1 のファイルに 15 バイトのデータを書き込んでいます。

```
VarString$=STRING$ (15, "X")
PUT #1,,VarString$
```

recordnumber&, *variable*, またはその両方を省略することができます。*recordnumber&* を省略して *variable* を指定するときは、カンマを入力しなければなりません。

```
GET #4,,FileBuffer
PUT #4,,FileBuffer
```

両方の引数を省略する場合は、カンマは必要ありません。

```
GET #4
PUT #4
```

GET, PUT ステートメントを使うと、Visual Basic の通信ファイルに対して、固定長の入出力を行うことができます。**GET** ステートメントは、通信エラーが起こると無制限に待つ可能性があります。このため、**PUT, GET** ステートメントを使って通信を行うときは、十分に注意してください。

GET ステートメントを **FIELD** ステートメントと一緒に使うとき、**GET** ステートメントを実行したあとで **INPUT #** や **LINE INPUT #** ステートメントを使うことによって、ランダムアクセス ファイルのバッファから文字列を読み込むことができます。また、**GET** ステートメントのあとで **EOF** 関数を使うと、**GET** ステートメントがファイルの終わりに達したかどうかを調べることができます。

FIELD ステートメントによって定義されたファイル バッファを使うとき、**PUT** ステートメントを実行する前に、**LSET, RSET, PRINT #, PRINT # USING, WRITE #** ステートメントを使って、ランダムファイルのバッファに文字データを書き込みます。**WRITE #** ステートメントを実行した場合、バッファ中のキャリッジ リターンの位置までスペースで埋められます。バッファのサイズを超えて読み書きを行おうとすると、エラーになります。

参照　**GET** ステートメント (グラフィックス), **PUT** ステートメント (グラフィックス)

GET, PUT ステートメント (グラフィックス)

機能

GET ステートメントは、画面上のグラフィックス イメージを配列に保存します。**PUT** ステートメントは、**GET** ステートメントで保存したグラフィックス イメージを、画面に表示します。フォームが表示されている状態では利用できません。

構文

GET [**STEP**](*x1!,y1!*)-[**STEP**](*x2!,y2!*), *arrayname*[(*index%*)]

PUT [**STEP**](*x1!,y1!*), *arrayname*[(*index%*)][, *actionverb*]

STEP	座標を相対座標で指定することを示します。
(*x1!,y1!*)	**GET** ステートメントでは保存するグラフィックス イメージの左上のビューポート座標、**PUT** ステートメントでは表示するグラフィックス イメージの左上のビューポート座標をそれぞれ示す数式。
(*x2!,y2!*)	保存するグラフィックス イメージの右下のビューポート座標を示す数式。
arrayname	イメージを保存する配列。この配列は、どのような数値型でもかまいませんが、イメージ全体を格納できるだけのサイズを持っていなければなりません。
index%	イメージを保存する、配列内の最初の要素を示す数式。
actionverb	これから表示するイメージと、そのとき画面上に表示されているイメージとの関係を示す値。この引数を使うと、特殊な効果を使ってイメージを表示できます。デフォルトは **XOR** です。

解説

フォームが表示されているときに **GET** および **PUT** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **GET** または **PUT** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

ビューポート座標は、**WINDOW** ステートメントが実行されていなければ物理座標と同じです。物理座標は、**VIEW** ステートメントが実行されていなければ、画面左上角を原点とする絶対座標です。

STEP キーワードは、現在のグラフィックス出力位置 (直前にプロットされた点) からの相対座標であることを示します。例えば、グラフィックス出力位置が (10,10) のときに STEP (4, 5) と指定すると、座標 (10+4, 10+5) すなわち (14,15) を意味します。

GET ステートメントは、対角の座標で指定した画面上の四角形のイメージを、*arrayname* で指定した配列に転送します。反対に **PUT** ステートメントは、配列に格納したイメージを画面上に転送します。

四角形のイメージを保存するために必要な配列の大きさ (バイト単位) は、次の式によって求めることができます。

```
4 + INT(((x2 - x1 + 1) +7)/8) * planes * ((y2 - y1) + 1)
```

配列の 1 要素当りのバイト数は次のとおりです。

- 整数配列の要素は 2 バイト
- 長整数配列の要素は 4 バイト
- 単精度配列の要素は 4 バイト
- 倍精度配列の要素は 8 バイト
- 通貨型配列の要素は 8 バイト

例えば、1 ピクセルを 4 プレーンで (1 色を 4 ビットで) 表現するスクリーン モードで、**GET** ステートメントを使ってイメージを保存するとします。イメージの左上隅の座標が (0,0) で、右下隅の座標が (32,32) であると、配列に必要なバイト数は、4 + INT((33 + 7)/8) * 4 * (33) つまり 660 になります。これは、330 個の要素からなる整数配列があると、イメージを格納するのに十分であることを意味します。

配列のデータ型が整数型か長整数型でなければ、**GET** ステートメントを実行したあとで、配列の内容を直接調べても意味がありません。グラフィックス イメージを含む非整数型の配列をチェックしたり操作すると、ランタイム エラーの原因になります。

GET ステートメントと **PUT** ステートメントを組み合わせて使うと、アニメーションを作成することができます。

PUT ステートメントの *actionverb* 引数に指定できる値は次のとおりです。

キーワード	機能
PSET	イメージをそのまま画面上に転送します。イメージの各点は、**GET** ステートメントで画面から読み出したときと同じ色属性を持ちます。
PRESET	**PSET** と同様ですが、イメージを反転 (たとえば白ならば黒) して画面上に転送します。
AND	イメージを画面上のイメージに重ね合わせます。画面には、格納されているイメージと画面上のイメージの間で、 AND 論理演算を行った結果のイメージが表示されます。このとき、格納されているイメージの点が、画面上のイメージの点と同じ色の場合にだけ、画面上にそのまま表示されます。
OR	イメージを画面上のイメージと合成します。画面上のイメージは、格納されているイメージによって消去されません。画面には、格納されているイメージと画面上のイメージの間で、 OR 論理演算を行った結果のイメージが表示されます。
XOR	アニメーションなどで使われる特殊なモードです。配列中のイメージの各点に対応する画面上の点が反転されます。これは、カーソルと同様な処理を行います。画面にあるイメージを表示したあとで、再び同じイメージを表示すると、画面のイメージが元に戻ります。このモードを使うと、画面上の背景を消さずに、あるオブジェクトを動かすことができます。 **XOR** が、省略した場合のデフォルトです。

GET ステートメントでは，(*x1!,y1!*) および (*x2!,y2!*) 引数で、保存するイメージの範囲 (四角形) の座標を指定します。この範囲は、画面内に収まっていなければなりません。**PUT** ステートメントでは、(*x1!,y1!*) 引数で、表示する範囲 (四角形) の左上の点の座標を指定します。この範囲全体が、そのときのビューポート内に収まっている必要があります。

1 組の **GET** ステートメントと **PUT** ステートメントは、同じスクリーン モードで実行する必要があります。

参照　**GET** ステートメント (ファイル I/O)，**PUT** ステートメント (ファイル I/O)

GetFEPMode% ルーチン

機能　日本語入力システムの状態を取得するルーチンです。

構文　**GetFEPMode%**

解説　このルーチンをプログラミング環境上で利用するには、何らかの Quick ライブラリをロードしている必要があります。VBDOS 起動時に /L オプションを指定してください。また、**DECLARE** ステートメントで FUNCTION プロシージャとして宣言されている必要があります。次のように宣言します。

```
DECLARE FUNCTION GetFEPMode%
```

この宣言は、インクルード ファイル VBDOS.BI にも含まれています。

GetFEPMode% ルーチンは、日本語入力システムがオンになっているときは -1、オフになっているときは 0 を返します。

日本語入力システムの状態を設定するには、SetFEPMode ルーチンを使います。

フォームが表示されている場合、日本語入力システムをオンにできるのは、フォーカスがテキスト ボックスか、Style プロパティが 0 または 1 のコンボ ボックスにあるときだけです。それ以外のコントロールまたはフォームにフォーカスがあるときに SetFEPMode ルーチンを呼び出すと、次にフォーカスがテキスト ボックスまたはコンボ ボックスに移ったときに、日本語入力システムがオンまたはオフになります。

フォーカスがテキスト ボックスまたはコンボ ボックスにあるときに GetFEPMode% ルーチンを呼び出すと、そのときの日本語入力システムの状態を示す値を返します。それ以外のコントロールまたはフォームにフォーカスがあるときに呼び出すと、テキスト ボックスまたはコンボ ボックスに次にフォーカスが移ったときに、日本語入力システムがどちらの状態になるかを示す値を返します。

日本語入力システムがインストールされていないときに GetFEPMode% ルーチンを呼び出すと、常に 0 を返します。

日本語入力システムは、MS-KANJI API に準拠していなければなりません。

参照　SetFEPMode ルーチン，**SETFOCUS** メソッド，テキスト ボックス，コンボ ボックス，**$INCLUDE** メタコマンド

GETTEXT メソッド

機能　**CLIPBOARD** オブジェクトに格納されているテキスト文字列を返します。

構文　**CLIPBOARD.GETTEXT()**

対象　**CLIPBOARD** オブジェクト

解説　**GETTEXT** メソッドは、関数のように値を返します。次のように文字列式の中で使うことに注意してください。

```
A$ = CLIPBOARD.GETTEXT()
```

CLIPBOARD オブジェクトとそれに適用可能なメソッドを使うと、コード中から **CLIPBOARD** オブジェクトの内容を操作できます。テキスト ボックスやコンボ ボックスの SelStart, SelLength, SelText プロパティと組み合わせて使うことによって、メニューやコマンド ボタンでカット & ペーストの機能を実現できます。

CLIPBOARD オブジェクトの内容が空のときは、null 文字列 ("") が返されます。

参照　**CLEAR** メソッド，**SETTEXT** メソッド

GOSUB...RETURN ステートメント

機能　サブルーチンへの分岐と復帰を行うフロー制御ステートメントです。

構文　**GOSUB** *line-label-to*

.

.

.

RETURN [*line-label-resume*]

line-label-to　サブルーチンの先頭を示す行番号または行ラベル。

line-label-resume　サブルーチンの復帰先を示す行番号または行ラベル。省略すると、**GOSUB** キーワードの次のステートメントから実行が続けられます。

解説

GUSUB ステートメントで呼び出すのはサブルーチンであり、**SUB** ステートメントで定義される SUB プロシージャでないことに注意してください。SUB プロシージャは通常、**CALL** ステートメントで呼び出します。

RETURN に対して行番号や行ラベルを指定しないと、サブルーチン終了後、**GOSUB** ステートメントの直後のステートメントから実行が続けられます。指定すると、その行ラベルや行番号が示す行に制御が移ります。ただしこの行 (サブルーチンの復帰先) はモジュール レベル コードでなければならず、プロシージャの中に制御を戻すことはできません。戻り先を指定した **RETURN** を使うと、プログラムの理解やデバッグが困難になるので、注意して使用してください。

サブルーチンは、プログラム中で何回でも呼び出すことができます。また、サブルーチンの内部から、別のサブルーチンを呼び出すこともできます。ネストの深さは、使用可能なスタック領域に制限されます (スタック領域を増やすには、**STACK** や **CLEAR** ステートメントを使います)。

自分自身を呼び出すサブルーチン (再帰的サブルーチン) を使うと、スタック領域が不足しやすくなります。

1 つのサブルーチンの中に、複数の **RETURN** を記述してもかまいません。

サブルーチンは、プログラム中のどこに置いてもかまいませんが、通常のプログラムの流れと明確に区別できるようにしてください。プログラムの実行が誤ってサブルーチンに入ってしまわないようにするには、サブルーチンの前に、実行を制御するための **STOP, END, GOTO** などのステートメントを挿入します。

GOSUB...RETURN の代わりに SUB および FUNCTION プロシージャを使うと、より構造化されたプログラムを作ることができます。プロシージャを使うことをおすすめします。

参照

CALL ステートメント (Basic プロシージャ), **FUNCTION** ステートメント, **SUB** ステートメント

GotFocus イベント

機能

オブジェクトがフォーカスを得たときに発生します。

構文

SUB Form_GotFocus ()

SUB *ctlname*_**GotFocus** ([*Index* **AS INTEGER**])

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，水平スクロール バー，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

フォーカスは、クリックや [TAB] キーを押すなどのユーザーの操作が行われたときや、コード中で **SETFOCUS** メソッドが実行されたときに移動します。

フォームは、コントロールが配置されていないか、配置されているすべてのコントロールが無効または非表示に設定されているときだけ、フォーカスを持つことができます。

GotFocus イベント プロシージャには、通常、コントロールまたはフォームが最初にフォーカスを得たときの処理を記述します。例えば、フォーム上のそれぞれコントロールの機能に関する説明を、画面やステータス バーに表示するときに使います。また、あるコントロールがフォーカスを得たときに、その他のコントロールを無効または有効にしたり、別のコントロールを新たに加えたりすることができます。

Enabled および Visible プロパティが True (-1) に設定されているオブジェクトだけが、フォーカスを得ることがでます。

コントロール間のフォーカスの移動をキーボードから操作できるようにするには、タブ オーダーを設定するか、フォーム上のコントロールにアクセス キーを割り当てます。

すでにフォーカスを持っているコントロールに対して **SETFOCUS** メソッドが実行されても、2 回目の GotFocus イベントは発生しません。

コントロールがフォーカスを得たり失ったりするたびに、コントロールの色を変更すると、視覚的に分かりやすいユーザー インターフェイスになります。

参照

ActiveControl プロパティ，ActiveForm プロパティ，Enabled プロパティ，LostFocus イベント，**SETFOCUS** メソッド，TabIndex プロパティ，TabStop プロパティ，Visible プロパティ

GOTO ステートメント

機能

指定した行へ無条件に分岐するフロー制御ステートメントです。

構文

GOTO *label-line*

label-line 制御を移す行を示す行番号または行ラベル。プログラムの実行は、その行から続きます。

解説

GOTO ステートメントは、行番号または行ラベルで指定した行に無条件で分岐します。**GOTO** ステートメントの分岐先は、分岐元と同じレベルの行でなければなりません。

GOTO ステートメントを多用すると、プログラムは読みづらくなり、デバッグが難しくなります。構造化された制御ステートメントを使うことをおすすめします。

GOTO ステートメントは、SUB, FUNCTION, 複数行にわたる **DEF FN** 関数などへ入るまたは出るために使うことはできませんが、それらの内部でのプログラムの実行の流れを制御するために利用することはできます。

GOTO は、**ON ERROR** ステートメントでもキーワードとして使われます。

参照

DO...LOOP ステートメント，**FUNCTION** ステートメント，**IF...END IF** ステートメント，**ON ERROR** ステートメント，**SELECT CASE** ステートメント，**SUB** ステートメント

Height, Width プロパティ

機能

オブジェクトの高さ (Height)、幅 (Width) を決めます。

構文

{[*formname*.][*ctlname*.]|**SCREEN**.}**Height**[= *height%*]

{[*formname*.][*ctlname*.]|**SCREEN**.}**Width**[= *width%*]

height% オブジェクトの高さを示す値。

width% オブジェクトの幅を示す値。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，**SCREEN** オブジェクト，テキスト ボックス，垂直スクロール バー

解説

Height と Width プロパティの値は、常にキャラクタ単位で表されます。

オブジェクトの高さと幅とは、次のサイズを示します。

オブジェクト	示すサイズ
コントロール	外側の高さと幅。ドライブ リスト ボックスと、Style プロパティの値が 0 (ﾄﾞﾛｯﾌﾟﾀﾞｳﾝｺﾝﾎﾞ) または 2 (ﾄﾞﾛｯﾌﾟﾀﾞｳﾝﾘｽﾄ) のコンボボックスの高さは、1 に固定されています。
フォーム	外側の高さと幅。境界とタイトル バーを含みます。
SCREEN	画面全体の高さと幅。デザイン時は使用できず、また実行時は読み込み専用です。

Height プロパティに対して有効な値の範囲は次のとおりです。

オブジェクト	設定可能な値
コントロール	1～254
フォーム	2～画面の行数 (20 または 25)
SCREEN	20 または 25 (読み込み専用)

Width プロパティに対して有効な値の範囲は次のとおりです。

オブジェクト	設定可能な値
コントロール	1～254
フォーム	2～画面の桁数 (80)
SCREEN	80 (読み込み専用)

フォームやコントロールでは、ユーザーやコード中からの操作によってオブジェクトのサイズが変更されると、これらのプロパティの値が変化します。

Height および Width プロパティに 0 以下の値を設定しようとするとエラーが発生します。フォームの Height および Width プロパティでは、BorderStyle や MixButton プロパティなどの設定によって、設定できる最小値は変化します。1 以上で、そのとき設定できる最小値より小さい値を設定しようとすると、そのときの最小値になります。例えば BorderStyle プロパティが 0 以外の場合、境界に 1 行分が使われるため、設定できる最小値は 3 になります。このとき、1 や 2 に設定しようとしても 3 と見なされます。

コントロールの高さと幅は、1～254 の範囲で設定できます。ただし、フォームの高さや幅を超える部分は画面には表示されません。

Height および Width プロパティや、関連するプロパティである Left および Top プロパティは、オブジェクトのサイズ変更や移動などの、オブジェクト自身の位置や領域の計算が必要なときに使います。これに対して ScaleHeight および ScaleWidth プロパティは、文字の出力や子コントロールの移動など、オブジェクト内部の領域の計算が必要なときに使います。

フォームを使わないアプリケーションでは、**WIDTH** ステートメントを使ってください。

参照　**WIDTH** ステートメント

HEX$, OCT$ 関数

機能　**HEX$** 関数は、引数に指定した値を 16 進数で表現した文字列を返す、文字列処理関数です。**OCT$** 関数は、引数に指定した値を 8 進数で表現した文字列を返す、文字列処理関数です。

構文　**HEX$**(*numeric-expression&*)

OCT$(*numeric-expression&*)

numeric-expression&　変換元の値を示す数式。

解説　引数に小数点を持つ数値を指定すると、整数型または長整数型に変換されてから評価されます。

Hidden プロパティ

Archive プロパティを参照してください。

HIDE メソッド

機能　フォームの Visible プロパティを False (0) に設定し、非表示にします。

構文　[{*formname*.|**SCREEN.**}]**HIDE**

対象　フォーム，**SCREEN** オブジェクト

解説　**HIDE** メソッドは、フォームの Visible プロパティを False (0) に設定するのと同じ効果があります。

SCREEN オブジェクトに対して **HIDE** メソッドを使うと、そのとき表示されていたすべてのフォームの Visible プロパティが False (0) に設定されます。これらのフォームを再び表示するには、**SCREEN.SHOW** を使います。

```
SCREEN.HIDE
SCREEN.SHOW
```

CIRCLE, LINE などのグラフィックス ステートメントを使うときは、その前にすべてのフォームを非表示にする必要があります。

非表示になっているフォームやそれ上のコントロールは、ユーザーからは操作できません。ただし、実行中のプログラムや Timer イベントに対するイベント プロシージャからは操作可能です。

アンロードされているフォームに対して **HIDE** メソッドを使うと、そのフォームを非表示のままロードすることができます。フォームやそれ上のコントロールのプロパティは、ロード時に初期化されます。

フォームを非表示にすると、使用メモリを節約するために、表示だけに関連するコードは自動的にメモリからクリアされます。ただしプロパティなど、プログラム中から操作可能な部分はメモリに残ります。

参照 **SHOW** メソッド，**UNLOAD** ステートメント，Visible プロパティ

HOUR 関数

機能 日付と時刻のシリアル値を評価して、0(12:00 A.M.)～23(11:00 P.M.) までの範囲で、何時かを返します。

構文 **HOUR**(*serial*)

serial 日付と時刻を表すシリアル値 (-53,688～65,380)。

IF TYPEOF ステートメント

機能

コントロールの種類によって、条件分岐を行うフロー制御ステートメントです。

構文

IF {*condition*|**TYPEOF** *ctlname* **IS** *controltype*} **THEN**

[*statementblock-1*]

[**ELSEIF TYPEOF** *ctlname* **IS** *controltype* **THEN**

[*statementblock-2*]...]

[**ELSE**

[*statementblock-n*]]

END IF

condition	真 (0 以外の値) または偽 (0) を返す式。
ctlname	コントロールの名前。
controltype	コントロールの種類を示す文字列。
statementblock-1	任意の行数のステートメント。
...	
statementblock-n	

解説

controltype 引数には、コントロールの種類を示す、次のいずれかの名前を指定します。

指定する名前	コントロールの種類
Label	ラベル
CommandButton	コマンド ボタン
OptionButton	オプション ボタン
CheckBox	チェック ボックス
TextBox	テキスト ボックス
ListBox	リスト ボックス
ComboBox	コンボ ボックス
PictureBox	ピクチャー ボックス
Frame	フレーム
Timer	タイマー
HScrollBar	水平スクロール バー
VScrollBar	垂直スクロール バー
DirListBox	ディレクトリ リスト ボックス
DriveListBox	ドライブ リスト ボックス
FileListBox	ファイル リスト ボックス
Menu	メニュー

IF TYPEOF ステートメンの機能は、条件がコントロールの種類であることを除き、**IF...END IF** ステートメントと同じです。

ELSE や **ELSEIF** ブロックは、省略可能です。また、**ELSEIF** ブロックはいくつ存在してもかまいません。さらに、*statementblock* 中に別のブロック形式の **IF** ステートメントをネストして置くこともできます。

参照　**IF...END IF** ステートメント，**SELECT CASE** ステートメント

IF...END IF ステートメント

機能　指定した条件によって、条件付きの実行や分岐を行うフロー制御ステートメントです。

構文

ブロック形式

IF *condition-1* **THEN**

　[*statementblock-1*]

[**ELSEIF** *condition-2* **THEN**

　[*statementblock-2*]...]

[**ELSE**

　[*statementblock-n*]]

END IF

行形式

IF *condition* **THEN** *statements* [**ELSE** *statements*]

condition	真 (0 以外の値) または偽 (0) を返す式。
statementblock-1	任意の行数のステートメント。
...	
statementblock-n	
statements	1 つまたは複数のステートメント。複数のステートメントを指定するときは、コロン (:) でつなげる必要があります。

解説　コントロールの種類を条件にしたいときは、**IF TYPEOF** ステートメントを使ってください。

条件が単純で、それに対応する動作が 1 文しかない場合は、行形式の **IF** ステートメントが便利です。

ブロック形式の **IF** ステートメントには、次のような利点があります。

- ブロック単位で条件分岐を設定できるので、より構造化された柔軟なプログラムを作成できます。
- 1 つの構造の中で、分岐条件を組み合わせて記述できます。
- 1 行に記述する必要がないため、より長いステートメントや構造を記述できます。
- 論理構造にしたがって記述できるため、デバッグや変更が容易になります。
- ブロック形式の **IF** を実行すると、まず最初の論理式 (*condition-1*) が評価されます。
- 式の値が真 (0 以外の値) なら、**THEN** に続くステートメント (*statementblock-1*) が実行されます。
- 偽(0) なら、以降の **ELSEIF** に指定された条件式 (*condition-2* など) を順番に評価します。

結果が真になる条件式が見つかると、対応する **THEN** に続くステートメント (*statementblock-2* など) が実行されます。**ELSEIF** で指定された条件式がいずれも偽であると、**ELSE** ステートメントに続くステートメント (*statementblock-n*) が実行されます。**THEN** または **ELSE** に続くステートメントの実行が終わると、**IF...THEN...ELSE** ステートメントは終了し、**END IF** の次のステートメントからプログラムの実行が続けられます。

ELSE や **ELSEIF** ブロックは、省略可能です。また、**ELSEIF** ブロックはいくつ存在してもかまいません。さらに、*statementblock* 中に別のブロック形式の **IF** ステートメントをネストして置くこともできます。

THEN キーワードの後ろに記述があるかどうかで、その **IF** ステートメントがブロック形式であるかどうかが判断されます。コメント以外が **THEN** の後に続いていれば、行形式の **IF** ステートメントと判断されます。

ブロック **IF** ステートメントは、行の最初のステートメントでなければなりません。また、**ELSE, ELSEIF, END IF** の前には、行番号か行ラベルのみを置くことができます。ブロックの終りは、**END IF** ステートメントでなければなりません。

statementblock には、行番号や行ラベルを指定した **GOTO** を置くことができます。行番号の前の **GOTO** は省略可能ですが、行ラベルの前の **GOTO** は省略できません。

行形式を必ず使わなければならない状況はありません。行形式の **IF...ENDIF** ステートメントは、常にブロック形式で置き換えることができます。

行形式でも、1 つの条件に対して複数のステートメントを記述できますが、次に示すように、すべてのステートメントをコロンで区切って同じ行に置かなければなりません。

```
IF A > 10 THEN A = A + 1: B = B + A: LOCATE 10,22: PRINT B,A
```

参照　**IF TYPEOF** ステートメント，**SELECT CASE** ステートメント

$INCLUDE メタコマンド

機能

指定したファイルをインクルードするようにコンパイラ (BC.EXE) またはプログラミング環境 (VBDOS.EXE) に指示するメタコマンドです。

構文

{'|**REM**} **$INCLUDE**: '*filespec*'

'*filespec*'	インクルードする Basic ファイル名を示す文字列。必ずシングルクォーテーション (') で囲まなければなりません。

解説

すべてのメタコマンドは、行頭にシングル クォーテーション (') または **REM** を付け、コメント行として記述します。また、**$INCLUDE** の後には必ずコロン (:) を付けなければならないことに注意してください。

コンパイラ (BC.EXE) は、**$INCLUDE** メタコマンドを次のように処理します。

1. Basic ソース ファイル中で **$INCLUDE** メタコマンドを見つけると、そのソース ファイルの処理を一時中断します。
2. *filespec* で指定されたインクルード ファイルを探し、内容を読み込んで処理します。
3. インクルード ファイルの処理が終了すると、ソース ファイル中の **$INCLUDE** の次の行から、処理を続けます。

インクルード ファイルは通常、**$FORM** メタコマンド，宣言，定数などをグループ化するために使います。宣言を行う部分を 1 つのインクルード ファイルにまとめると、プロジェクト中に複数の宣言部分を作るより、更新などの管理が簡単になります。

プログラミング環境でインクルード ファイルを作成したときは、必ずバイナリ形式ではなく、テキスト形式でセーブしてください。

インクルード ファイルは、プロジェクト ファイルに記録する必要はありません。文字列検索などを行うために、プロジェクトにインクルード ファイルを加えるには、[ファイル追加] コマンドを選択して、表示されるダイアログ ボックス中のファイル形式の欄から、"インクルード[*.BI]" を選択する必要があります。

インクルード ファイルの内容を表示するには、[表示] メニューの [インクルード ファイル表示] または [インクルード ファイル編集] コマンドを選択します. [インクルード ファイル表示] コマンドでは、**$INCLUDE** メタコマンドの代わりにインクルード ファイルの内容が表示されますが、内容を変更することはできません。[インクルード ファイル編集] コマンドは、カレントのコード ウィンドウにインクルード ファイルの内容を表示します。

プログラミング環境上でプログラムを実行する場合、インクルード ファイルの中で **SUB** または **FUNCTION** ステートメントを使うことはできません. Visual Basic コンパイラ (BC.EXE) を使ってプログラムをコンパイルするときは、インクルード ファイルの中で **SUB** または **FUNCTION** ステートメントを使うことができます。通常は、イン

クルード ファイル中には宣言だけを含め、実行可能ステートメントは置かないようにします。

参照　**REM** ステートメント

Index プロパティ

機能　コントロール配列中で 1 つのコントロールを識別するインデックス番号を決めます。実行時には読み込み専用です。

構文　[*formname.*]*ctlname*[(*arrayindex%*)].**Index**

arrayindex%　インデックス番号を示す 0～32767 の整数。

対象　チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，テキスト ボックス，タイマー，垂直スクロール バー

解説　コントロール配列の要素はすべて同じコントロール名 (CtlName プロパティの値) を持つため、配列中の各コントロールを個別に参照するには Index プロパティの値を使います。Index プロパティの値は、次のようにコントロール配列の名前の後に () で囲んで指定します。

```
MyButton(3)
```

コントロール配列になっていないコントロールでは、Index プロパティは存在しません。参照しようとするとエラーになります。

コントロール配列中のコントロールがイベントの発生を認識すると、配列に対するイベント プロシージャが呼ばれて、Index プロパティの値が引数として引き渡されます。

デザイン時にコントロール配列を作成する場合、同じ CtlName のコントロールを生成するたびに、重複のない最小の番号が Index プロパティの値として割り振られて行きます。Index プロパティの最小値は 0 で、最初に生成されたコントロールに割り振られます。

実行時に **LOAD** ステートメントを使い、コントロール配列に新しい要素を追加する場合、添字として指定した値が Index プロパティの値になります。

デザイン時にコントロール配列からコントロールを削除する場合は、そのコントロールの CtlName プロパティを変更します。実行時に削除するときは **UNLOAD** ステートメントを実行します。

参照　CtlName プロパティ，Tag プロパティ

INKEY$ 関数

機能

標準入力デバイスから読み込んだ文字を返します。フォームが表示されている状態では利用できません。

構文

INKEY$

解説

フォームが表示されているときに **INKEY$** 関数を参照すると、"フォームが表示されているときは無効です" エラーが発生します。

INKEY$ 関数は、標準入力デバイス (通常はキーボード) から読み込んだ文字を表す、1 バイトまたは 2 バイトの文字列を返します。

INKEY$ 関数が返す値には、次の 4 種類があります。

- 通常のキーが押されたときは ASCII コード。
- 2 バイト文字が入力されたときは漢字連続コード。
- [BS], [PAGE DOWN] キーなどの特殊キーが押されたときは特殊キー コード。
- 何も入力されなければ null 文字列。

特殊キー コードについては、「機種依存情報」を参照してください。

漢字連続コードは、JIS 漢字コード (またはシフト JIS コード) の先頭の文字である 2 バイトのスペースを 10 進数の 256 として、JIS 漢字コード順にすき間なく並べて行ったコードです。各文字の漢字連続コードは、**ASC** 関数で求めることができます。

INKEY$ 関数の返す値は画面に表示されません。

[CAPS LOCK], [漢字] キーなどは **INKEY$** 関数に影響しません。また、[STOP] キーが押された場合、プログラミング環境で実行している場合や /D オプションを付けてコンパイルした場合は、プログラムの実行は停止します。/D オプションを付けずにコンパイルしたときは、対応する特殊キー コードが返されます。

KEY ステートメントを使って文字列をファンクション キーに割り当てた場合、**INKEY$** 関数 が入力待ち状態のときにそのファンクション キーを押すと、ファンクション キーに割り当てた文字列がプログラムに渡されます。キー トラッピングが有効なときは、**INKEY$** 関数よりキー トラッピングが優先されます。

INP 関数, OUT ステートメント

機能

INP 関数は、I/O ポートから読み込んだ 1 バイトのデータを返します。**OUT** ステートメントは、I/O ポートに 1 バイトのデータを送ります。

構文

INP(*port%*)

OUT *port%*, *data%*

port%	ハードウェアの I/O ポートを示す、0～65535 の整数。
data%	ポートに出力する、0～255 の整数。

解説

OUT ステートメントと **INP** 関数は、I/O ポートを通じてシステムのハードウェアを直接制御するためのものです。ハードウェアを直接操作するので、十分注意して使ってください。

参照

WAIT ステートメント

INPUT, LINE INPUT ステートメント

機能

INPUT ステートメントは、プログラムの実行時に、キーボードまたはファイルから入力されたデータを読み込み、それを変数に代入します。**LINE INPUT** ステートメントは、行全体 (255 文字まで) を、区切り文字を使わずに文字列変数に代入します。フォームが表示されている状態では利用できません。**INPUTBOX$** 関数を使ってください。

構文

INPUT[;] ["*prompt*"{;|,}] *variablelist*

LINE INPUT[;] ["*prompt*"{;|,}] *variablelist*

;	プログラムのユーザーがデータを入力したあとで [ENTER] を押しても、カーソルを次の行に移動しないようにすることを示します。
prompt	プログラムのユーザーにプロンプトとして表示するメッセージ。省略すると、疑問符 (?) だけが表示されます。また、*prompt* 引数の後ろにセミコロン (;) を指定すると、メッセージの後に疑問符が表示され、カンマ (,) を指定すると表示されません。
variablelist	入力された値を代入するための変数。複数の変数を指定するときは、カンマで区切って指定します。指定できる変数は、単純数値変数，単純文字列変数，数値配列の要素，文字列配列の要素，ユーザー定義型の要素のいずれかです。

解説

フォームが表示されているときに **INPUT** または **LINE INPUT** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

入力したデータは *variablelist* で指定した変数に代入されます。入力するデータ項目の数は、指定されている変数の数と等しくなければなりません。**INPUT** ステートメントでは、カンマの後ろにスペース，キャリッジ リターン，ライン フィード以外の文字が現れると、新しいデータ項目の始まりと見なされます。**LINE INPUT** ステートメントでは、[ENTER] が押されるまでのすべてのデータ (ただし 255 文字以内) が、指定した変数に代入されます。

入力した各データ項目のデータ型は、対応する変数のデータ型と一致していなければなりません。入力したデータの数または型がリスト中の変数と異なると、入力した値は無視され、"再入力してください" というメッセージが表示されます。正しいデータを入力するまで、値は変数に代入されません。

INPUT ステートメントに対して文字列を入力するとき、ダブル クォーテーション(") で囲む必要はありません。入力した最初の文字がダブル クォーテーションであると、そのダブル クォーテーションと次のダブル クォーテーションに囲まれた文字全体が、読み込む文字列であると見なされます。ダブル クォーテーションは、文字列の一部として扱われません。入力する文字列の最初の文字がダブル クォーテーションでなければ、次のカンマ，キャリッジ リターン，ライン フィードまでが 1 つの文字列として扱われます。

ユーザー定義型の変数に対して入力するときは、個々の要素ごとに指定してください。

```
TYPE Demograph
    FullName AS STRING * 25
    Age  AS INTEGER
END TYPE
DIM Person AS Demograph
INPUT "氏名と年齢を入力してください: ";Person.FullName, Person.Age
```

[ENTER] を押す前であれば、入力したデータを修正することができます。次の表は、入力行でのカーソルの移動，テキストの削除，テキストの挿入などを行うための組み合わせキーの一覧です。

キー操作	動作
[CTRL]+[¥] または [→]	カーソルを 1 文字分右に移動します。
[CTRL]+[]] または [←]	カーソルを 1 文字分左に移動します。
[CTRL]+[F] または [CTRL]+[→]	カーソルを 1 単語分右に移動します。
[CTRL]+[B] または [CTRL]+[←]	カーソルを 1 単語分左に移動します。
[CTRL]+[K] または [HOME]	カーソルを入力行の先頭に移動します。
[CTRL]+[N] または [END]	カーソルを入力行の末尾に移動します。
[CTRL]+[R] または [INS]	挿入モードの ON/OFF を切り換えます。挿入モードが ON になっているとき、文字を入力すると、カーソル位置から右側にある文字が右に移動します。

[CTRL]+[I] または [TAB]	カーソルの右側にタブを挿入するか (挿入モード ON)、カーソルの右側にタブを上書きします (挿入モード OFF)。
[DEL]	カーソル位置の文字を削除します。
[CTRL]+[H] または [BS]	カーソルがデータの先頭になければ、左側の文字を削除します。カーソルがデータの先頭にあると、カーソル位置の文字を削除します。
[CTRL]+[E] または [CTRL]+[END]	カーソルから行の終わりまでの文字をすべて削除します。
[CTRL]+[U] または [ESC]	カーソルの位置に関係なく、入力行全体を削除します。
[CTRL]+[M] または [ENTER]	入力行のデータを変数に格納します。
[CTRL]+[T]	画面最下行にあるファンクションキーの表示を ON/OFF します。
[STOP] または [CTRL]+[C]	データの入力を終了します (実行可能ファイルの場合は、プログラムを終了します)。

参照　**INKEY$** 関数，**INPUT$** 関数，**INPUT #** ステートメント，**LINE INPUT #** ステートメント，**INPUTBOX$** 関数，**LINE** ステートメント

INPUT #, LINE INPUT # ステートメント

機能　シーケンシャル デバイスやファイルからデータ項目を読み出して、それを変数に代入するファイル I/O ステートメントです。

構文

INPUT #*filenumber%*, *variablelist*

LINE INPUT #*filenumber%*, *variablelist*

filenumber%	オープンしているファイルのファイル番号。
variablelist	入力された値を代入するための変数。複数の変数を指定するときは、カンマで区切って指定します。指定できる変数は、単純数値変数，単純文字列変数，数値配列の要素，文字列配列の要素，ユーザー定義型の要素のいずれかです。

解説　フォームが表示されているときに KYBD: または CONS: に対して **INPUT #** または **LINE INPUT #** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

INPUT # ステートメントを使うと、ファイル中のデータ項目は **INPUT** ステートメントに対する入力データと同様に処理されます。数値データはスペース，キャリッジ リターン，ラインフィード，カンマで区切ります。文字列データはキャリッジ リターンまたはライン フィードで区切ります。データ項目の前のスペースは無視されます。ファイルの末尾 (EOF 文字) は、数値データと文字列データのいずれの場合でも、データ入力の終了を意味します。

LINE INPUT # ステートメントは、シーケンシャル ファイルの現在の位置から、キャリッジ リターンの直前までのすべての文字列を読み込みます。キャリッジ リターンとライン フィードは読み飛ばします。**LINE INPUT #** ステートメントは、データファイルの各行が複数のフィールドに分割されていたり、テキスト ファイルを 一度に 1 行ずつ読み込むときに便利です。

参照　**INKEY$** 関数，**INPUT** ステートメント，**INPUT$** 関数，**INPUTBOX$** 関数，**LINE INPUT** ステートメント，**LINE** ステートメント

INPUT$ 関数

機能　指定したファイルから、指定したバイト数の文字列を読み込んで返します。

構文　**INPUT$**(*n*[, [#]*filenumber%*])

n	ファイルから読み込むバイトを示す数式。
filenumber%	オープンしているファイルのファイル番号。省略すると、標準入力デバイスと見なされます。

解説　フォームが表示されているときに SCRN: または CONS: に対して **INPUT$** を実行すると、"フォームが表示されているときは無効です" エラーが発生します。

ファイルをランダムアクセス モードでオープンしたとき、引数 *n* の値は、**OPEN** ステートメントの **LEN** 節で設定したレコード長以下 (レコード長を設定していないときは 128 以下) でなければなりません。バイナリ モードかシーケンシャルアクセス モードでオープンしたとき、引数 *n* の値は 32,767 以下でなければなりません。

filenumber% を省略すると、文字は標準入力デバイス (通常はキーボード) から読み出されます。

MS-DOS のリダイレクトの機能 (<, >, >>) やパイプの機能 (|) を使うと、実行可能ファイルに対する標準入出力デバイスを変更できます。

INPUT# ステートメントと異なり、**INPUT$** 関数は読み込んだすべての文字を返します。ただし [STOP] を読み込んだ場合に限り、関数の実行が中断されます。

参照　**INPUT** ステートメント，**INPUT #** ステートメント，**LINE INPUT** ステートメント，**LINE INPUT #** ステートメント，**INPUT ¥** ステートメント

INPUT¥ 関数

機能　指定したファイルから、指定した文字数の文字列を読み込んで返します。

構文　**INPUT¥**(*n*[, [#]*filenumber%*])

n	ファイルから読み込む文字数を示す整数。
filenumber%	オープンしているファイルのファイル番号。省略すると、標準入力デバイスと見なされます。

解説　**INPUT¥** 関数の機能は、読み込む文字数を指定することを除き、**INPUT$** 関数と同じです。**INPUT$** 関数では、読み込むバイト数を指定します。

参照　**INPUT$** 関数，**INPUT** ステートメント，**INPUT #** ステートメント

INPUTBOX$ 関数

機能　指定したメッセージを含む、定形のダイアログ ボックスを表示し、ユーザーが入力した文字列を返します。グラフィックス モードでは実行できません。

構文　**INPUTBOX$**(*prompt$*[, *title$*[, *default$*[, *xpos%*, *ypos%*]]])

prompt$	ダイアログ ボックスに表示するメッセージを示す文字列式。最大で 255 文字まで指定できます。
title$	ダイアログ ボックスのタイトル バーに表示するタイトルを示す文字列式。省略するとタイトルは表示されません。 47 文字目以降は切り捨てられます。
default$	テキスト ボックス部分にデフォルトで表示するテキストを示す文字列式。省略すると何も表示されません。
xpos%	画面の左端からダイアログ ボックスの左端までの水平方向の距離を、文字数で示す数式。省略すると、ダイアログ ボックスは画面の中央に表示されます。
ypos%	画面の上端からダイアログ ボックスの上端までの垂直方向の距離を、文字数で示す数式。省略すると、ダイアログ ボックスは画面の中央に表示されます。

解説

<了解> ボタンが押されるか [ENTER] キーが押されると、テキスト ボックス部分の内容を返します。<取消> ボタンが押されたときや、コントロール メニューの [クローズ] コマンドでクローズされたときは、null 文字列 ("") を返します。

ダイアログ ボックスは、モーダルなダイアログ ボックスとして表示されます。このダイアログ ボックスが表示されている間は、他のフォームに対するイベントは発生しません。

このダイアログ ボックスは移動できますが、サイズは変更できません。また、このダイアログ ボックスの幅と高さは 46 x 16 に固定されています。

1 行に表示可能なメッセージは最大 30 バイトです。*prompt$* 引数に 30 バイトを超える長さの文字列を指定すると、スペースまたは 2 バイト文字の前後で自動的に改行されます。30 バイト以下の長さで意図的に改行したいときは、*prompt$* 引数に指定する文字列の途中に改行コード (CHR$(13) と CHR$(10)) を挿入してください。

タイトル バーに表示可能なタイトルは最大 40 バイトです。タイトルがこれより長いとき、超える部分は切り捨てられます。

xpos% と *ypos%* は、片方だけを指定することはできません。両方を省略すると、ダイアログ ボックスは画面の中央に表示されます。

参照

INPUT ステートメント，**INPUT #** ステートメント，**MSGBOX** 関数，**MSGBOX** ステートメント

INSTR 関数

機能

ある文字列の中から別の文字列を検索し、最初に見つかったバイト位置を返す文字列処理関数です。

構文

INSTR([*start%*,]*string-expression1$*, *string-expression2$*)

start%	*string-expression1$* の中で、検索を開始するバイト位置を示す、1 ～32,767 の整数。省略すると、*string-expression1$* の先頭と見なされます。
string-expression1$	検索する文字列。文字列変数、文字列式、リテラル文字列のいずれでもかまいません。
string-expression2$	検索対象となる文字列。文字列変数、文字列式、リテラル文字列のいずれでもかまいません。

解説

string-expression1$ の長さ (バイト単位) を調べるには **LEN** 関数を使います。

INSTR 関数が返す値は、次の条件によって異なります。

条件	戻り値
string-expression2$* が *string-expression1$ 内で見つかったとき	見つかった文字列の先頭位置
start% の長さが *string-expression1$* より長いとき	0
string-expression1$ がnull 文字列のとき	0
string-expression2$ が見つからないとき	0
string-expression2$ がnull 文字列のとき	*start%* (*start%* 指定している場合) 1 (*start%* を省略した場合)

参照　**KINSTR** 関数

INT 関数

FIX 関数を参照してください

Interrupt, InterruptX ルーチン

機能　Basic のプログラムから MS-DOS のシステム コールや BIOS コールなどを実行するためのルーチンです。

構文　**CALL Interrupt** (*interruptnum%, inregs, outregs*)

CALL InterruptX (*interruptnum%, inregs, outregs*)

interruptnum%	MS-DOS の割り込み番号を示す、0～255 の整数。
inregs	割り込みを行う前のレジスタの値を格納する変数。この変数は、Interrupt に対してはユーザー定義型である RegType 型、InterruptX に対しては RegTypeX 型でなければなりません。
outregs	割り込みを行ったあとのレジスタの値を格納する変数。この変数は、Interrupt に対してはユーザー定義型である RegType 型、InterruptX に対しては RegTypeX 型でなければなりません。

解説

Interrupt および InterruptX ルーチンは、以前のバージョンの QuickBasic, Microsoft Basic の INT86, INT86X ルーチンの代わりに使うことができます。Interrupt および InterruptX ルーチンを使うと、Visual Basic でより容易に MS-DOS の割り込みを行うプログラムを作成できます。**CALL** ステートメントを使って呼び出してください。

割り込みを行う前と後のレジスタの値は、ユーザー定義型 RegType および RegTypeX で宣言された変数によって引き渡されます。これらの型の各要素は、CPU のレジスタに対応しています。

Interrupt ルーチン用のユーザー定義型 RegType は、次のように定義されています (DS および ES レジスタは含まれていません)。

```
TYPE RegType
    AX AS INTEGER
    BX AS INTEGER
    CX AS INTEGER
    DX AS INTEGER
    BP AS INTEGER
    SI AS INTEGER
    DI AS INTEGER
    FLAGS AS INTEGER
END TYPE
```

InterruptX ルーチン用のユーザー定義型 RegTypeX は、次のように定義されています。

```
TYPE RegTypeX
    AX AS INTEGER
    BX AS INTEGER
    CX AS INTEGER
    DX AS INTEGER
    BP AS INTEGER
    SI AS INTEGER
    DI AS INTEGER
    FLAGS AS INTEGER
    DS AS INTEGER
    ES AS INTEGER
END TYPE
```

InterruptX ルーチンは、DS, ES レジスタの値を使います。これらのレジスタの現在の値を使うには、ユーザー定義型の要素を -1 に設定してください。

参照

Absolute ルーチン，**$INCLUDE** メタコマンド

Interval プロパティ

機能

Timer イベントが発生する時間間隔 (ms 単位) を決めます。

構文

[*formname.*]*ctlname*.**Interval**[= *milliseconds&*]

ctlname	タイマーの名前。
milliseconds&	Timer イベントの発生間隔を示す 0～65535 の整数。デフォルトは 0 で、Timer イベントは発生しません。

対象

タイマー

解説

Interval プロパティは、Timer イベントの発生間隔を設定します。0 に設定すると、そのタイマー コントロールに対する Timer イベントは発生しません。また、Enabled プロパティが、タイマーのオン/オフを決めています。したがって Timer イベントを発生させるには、Interval プロパティが 1 以上に設定されていると同時に、Enabled プロパティが True (-1) に設定されていなければなりません。

Interval プロパティは、デザイン時と実行時のどちらでも設定できます。

Interval プロパティには、ミリ秒単位で時間を設定しますが、実際のイベントの発生間隔はハードウェアに依存します。

参照

Timer イベント

IOCTL$ 関数, IOCTL ステートメント

機能

IOCTL$ 関数は、デバイス ドライバから現在のステータスを示す文字列を受け取り、その文字列を返します。**IOCTL** ステートメントは、デバイス ドライバへ制御文字列を送ります。

構文

IOCTL$([#]*filenumber%*)

IOCTL [#]*filenumber%*, *string$*

filenumber%	オープンしているデバイスのファイル番号。
string$	デバイス ドライバへ送るコマンド (制御文字列) を示す文字列式。

解説

コマンド (制御文字列) やステータスの内容はデバイス ドライバによって異なります。デバイス ドライバのマニュアルを参照してください。**IOCTL** 制御データ文字列の長さは、最大 32,767 バイトです。

IOCTL$ 関数は、通常次の目的で使います。

- **IOCTL** ステートメントが正常に実行されたかどうかを調べる。
- デバイスの現在のステータス情報を得る。

IOCTL ステートメントは、次の 3 つの条件が満たされるときだけ機能します。

- デバイス ドライバがインストールされている。
- デバイス ドライバが **IOCTL** ステートメントで送った文字列を処理できる。
- Visual Basic がそのデバイス上のファイルに対して **OPEN** ステートメントを実行し、そのファイルが現在もオープンされたままである。

ほとんどの MS-DOS のデバイス ドライバは、**IOCTL** 制御文字列を処理しません。このため、指定したデバイス ドライバが、このコマンドを受け取るかどうかを調べる必要があります。もしそのドライバが **IOCTL** 制御文字列を処理しない場合、"引数が有効範囲にありません" エラーになります。

Visual Basic のデバイス (LPT*n*, COM*n*, SCRN, CONS, PIPE) と MS-DOS のブロック デバイス (A:～Z:) は、**IOCTL** をサポートしません。

IOCTL$** 関数を使うと、通信デバイスを通じて、現在のボーレート，直前に起こったエラーの情報，論理行の幅などを知ることができます。**IOCTL$ 関数が返す値は、使っているデバイス ドライバによって異なります。

JIS$ 関数

機能　指定した文字列の、最初の文字の JIS コードを文字列として返します。

構文　**JIS$**(*string-expression$*)

string-expression$　調べたい文字列。

解説　*string-expression$* に指定した文字列の先頭 1 文字が 2 バイト文字なら、その文字の JIS コードを 16 進数で表した 4 バイトの文字列を返します。先頭の 1 文字が 1 バイト文字なら、その文字の ASCII コードを 10 進数で表した 3 バイト以下の文字列を返します。

参照　**ASC** 関数，**KTN$** 関数

KEXT$ 関数

機能　指定した文字列から、1バイト系または2バイト系文字だけを抜き出して返します。

構文　**KEXT$**(*string-expression$*, *func*)

string-expression$	元の文字列。
func	0または1。0を指定すると1バイト文字だけを抜き出し、1を指定すると2バイト文字だけを抜き出します。

解説　*string-expression$* の中に指定した文字種がない場合、null 文字列を返します。

参照　**ASC** 関数，**JIS$** 関数

KEY ステートメント (イベント トラッピング)

機能　指定したキー イベントのトラップを有効，無効，中断状態にします。また、ユーザー定義キーの設定も行います。フォームやコントロールに関連したイベントには影響を与えません。フォームが表示されている状態では利用できません。

構文　**KEY**(*keyno%*) {**ON**|**OFF**|**STOP**}

KEY *n%*, *string-expression$*

keyno%	ファンクション キー，方向キー，ユーザー定義キーのキー番号を示す整数。
ON	トラップを有効にすることを示します。 **ON KEY** ステートメントによる、ソフト キー，方向キー，ユーザー定義キーのイベントトラップが可能になります。
OFF	トラップを無効にすることを示します。このステートメントが実行されたあとに起こったイベントは、記憶されません。
STOP	トラップを中断します。このステートメントが実行されたあとでイベントが発生すると、そのイベントは記憶されます。次に **KEY**(*keyno%*) **ON** ステートメントが実行されると、すぐに **ON KEY** ステートメントでトラップされます。
n%	ユーザー定義キーに割り当てるキー番号を示す 15～25 の整数。
string-expression$	ユーザー定義キーとしてキー イベントのトラップに利用するキー。次の構文に従って文字列で指定します。 **CHR$**(*keyboardflag%*) + **CHR$**(*code%*)

解説

このステートメントで扱うイベントは、フォームやコントロールを使ったイベント ドリブン プログラミング モデルのイベントとは異なることに注意してください。**KEY** ステートメントは、フォームが表示されているときには使用できません。また、**KEY ON** ステートメントでキー イベントのトラップを有効にしてからフォームを表示しても、フォームが表示されている間はトラップは無効です。

カッコで囲んで指定する *keyno%* 引数には、トラップの対象となるファンクション キー，方向キー，ユーザー定義キーの番号を指定します。

値	キー
1～10	ファンクション キー [F1] ～ [F10]
11	[↑] キー
12	[←] キー
13	[→] キー
14	[↓] キー
15～25	ユーザー定義キー
26～31	ファンクション キー [F11] ～ [F16] (機種によっては無効です)

26～31 に割り当てられているファンクション キーは、ハードウェアによっては無効です。30, 31 は、QuickBasic や Microsoft Basic と意味が異なりますので注意してください。

イベント トラップが有効であるとき、そのイベントが発生していないかどうか (指定したキーが押されていないかどうか) が自動的に監視されています。このチェックは、プログラミング環境で実行されている場合と、/V オプション付きでコンパイルされた実行可能ファイルでは、ステートメントごとに行われます。/W オプション付きでコンパイルされた実行可能ファイルでは、行番号または行ラベルごとにチェックされます。イベントが発生していると、**ON KEY** ステートメントで設定されているイベント処理ルーチンに制御が移ります。

イベント トラップを行うステートメントを含むプログラムを BC コマンドラインからコンパイルするとき、/W または /V オプションを付けずにコンパイルすると、"ON event ステートメントを含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

KEY(*keyno%*) **STOP** ステートメントでトラップを中断している間に起きたイベントは、**KEY**(*keyno%*) **OFF** ステートメントが実行されると失われます。

イベントがトラップされ、対応する処理ルーチンに制御が移ると、再帰的なトラップが起こらないように自動的に **KEY**(*keyno%*) **STOP** が実行されます。処理ルーチンが終了すると、そのルーチンの中で明確に **KEY**(*keyno%*) **OFF** が実行されていない限り、自動的に **KEY**(*keyno%*) **ON** ステートメントが実行されます。

ソフト キーとしてファンクション キーに割り当てられている文字列が入力されても、イベントとは見なされません。

KEY ステートメントを次の形式で使うと、ファンクション キーや方向キー以外のキーやキーの組み合わせもトラップすることができます。

KEY *n%*, **CHR$**(*keyboardflag%*)+**CHR$**(*code%*)

引数 *n%* には、ユーザー定義キーを示す 15～25 の値が指定できます。この値が、そのキーまたはキーの組み合わせを示すキー番号になります。

code% には、元になるキーのキーボードスキャンコードを指定します。キーボード スキャンコードについては、『機種依存情報』を参照してください。

keyboardflag% には、組み合わせる特殊キーを示す、次のような値を指定します。複数の特殊キーの組み合わせを示すには、以下の値の合計を指定してください。

キー	*keyboardflag%* に指定する値
[SHIFT]	1
[CTRL]	2
[メニュー]	4
キーの組み合わせを無視	128

128 を指定すると、[SHIFT], [CRTL] などの上記の特殊キーが押されているかどうかをチェックしないようになります。これによって、どのようなキーの組み合わせで押されても、単独のキーが押されたときと同じように扱うことができます。

参照

KEY ステートメント (代入)，KeyDown イベント，KeyPress イベント，KeyUp イベント，**ON KEY** ステートメント

KEY ステートメント (代入)

機能

ファンクション キーに文字列を割り当てます。また、ファンクション キーの内容の表示行のオン/オフも行います。フォームが表示されている状態では利用できません。

構文

KEY {*key%*, *string-expression$*|**LIST**|**ON**|**OFF**}

key%	文字列を割り当てるファンクション キーを示す 1 以上の整数。[F1] キーが 1 になります。
string-expression$	ファンクションキーに割り当てる、 15 文字以内の文字列。対応するファンクションキーが押されると、その値が返されます。 15 文字を超える文字列を指定すると、 16 文字以降は無視されます。null 文字列を指定すると、そのファンクションキーは文字列を返しません。CR-LF やダブル クォーテーション (") などを含めるには、**CHR$** 関数を使ってください。
LIST	キーと、それに割り当てられている文字列を表示することを示します。
ON	ファンクション キーの内容の表示行を表示することを示します。設定されている各文字列の最初の 6 文字が、画面最下行に表示されます。
OFF	ファンクション キーの内容の表示行を消去することを示します。

解説

フォームが表示されているときに **KEY** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

KEY ステートメントは、ファンクション キーが押されたときに返る文字列を設定します。ファンクション キーで入力した文字列も、**INPUT$** 関数，**INPUT**ステートメント，**INKEY$** 関数を使って読み込むことができます。

n% に有効範囲外の値を指定すると、"引数が有効範囲にありません" エラーが発生します。このとき、前に設定されていた文字列がそのまま残ります。

参照

KEY ステートメント (イベント トラッピング)

KeyDown, KeyUp イベント

機能

ユーザーがキーを押したとき (KeyDown)、または離したとき (KeyUp) に、そのときフォーカスを持っているオブジェクトに対して発生します。

構文

SUB Form_KeyDown (*KeyCode* **AS INTEGER**, *Shift* **AS INTEGER**)

SUB *ctlname*_**KeyDown** ([*Index* **AS INTEGER**,]*KeyCode* **AS INTEGER**, *Shift* **AS INTEGER**)

SUB Form_KeyUp (*KeyCode* **AS INTEGER**, *Shift* **AS INTEGER**)

SUB *ctlname*_**KeyUp** ([*Index* **AS INTEGER**,]*KeyCode* **AS INTEGER**, *Shift* **AS INTEGER**)

KeyCode	操作されたキーを示すキー コードが渡されます。
Shift	イベント発生時の [SHIFT], [CTRL], [メニュー] の各キーの状態を示す値が渡されます。
Index	コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，水平スクロール バー，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

KeyDown および KeyUp イベントは、ユーザーが押した特殊キーに従って処理を記述するときに便利です。キーが押されると、そのときフォーカスを持っているオブジェクトに対してイベントが発生します。

KeyDown および KeyUp イベントはすべてのキーを検出できますが、おもに次のキーを検出するために使われます。また、キーが押されたときと離されたときの両方の処理を記述したいときにも使います。

- ファンクション キーなどの特殊キー
- 方向キーや [PAGE DOWN], [PAGE UP], [HOME], [END] などのナビゲーション キー
- [SHIFT], [CTRL], [メニュー] の各キーの組み合わせ

KeyCode 引数に指定した変数には、押されたキーを示すキー コードが格納されます。

画面にそのまま表示されるキー (アルファベット，数字，記号) が押されたときは、*KeyCode* 引数には ASCII コードが渡されます。このとき、大文字小文字は区別されますから注意してください。ただし、これらのキーだけをイベントとして認識するには、KeyPress イベントの方が便利です。

ファンクション キーなどの特殊キーが押されたときの処理を記述するには、インクルード ファイル CONSTANT.BI で定義されている定数を使って、押されたキーを確認してください。例えば、[F1] キーのキー コードは KEY_F1, [↓] キーは KEY_DOWN です。インクルード ファイルは、**$INCLUDE** メタコマンドを使って各モジュールにインクルードします。

Shift 引数に指定した変数には、イベント発生時の [SHIFT], [CTRL], [メニュー] の各キーの状態を示す値が格納されます。[SHIFT] キーを 1, [CTRL] キーを 2, [メニュー] キーを 4 として、イベント発生時に押されていたキーの値を合計した値が渡されます。例えば、値が 7 であれば 3 つのキーすべてがいっしょに押されていたことを示し、値が 3 なら [SHIFT] キーと [CTRL] キーがいっしょに押されていたことを示します。

アクセス キーとショートカット キーは、フォーム デザイナで簡単に設定できます。これらのキーを設定するために、KeyDown と KeyUp イベントを使う必要はありません。

次の状態のときは、KeyDown および KeyUp イベントは発生しません。

キー	状態
[ENTER]	Default プロパティが True (-1) に設定されているコマンド ボタンが存在するとき
[ESC]	Cancel プロパティが True (-1) に設定されているコマンド ボタンが存在するとき
[TAB]	2 つ以上のコントロールがフォーム上に存在するとき

QuickBasic や Microsoft Basic Professional Development System で用意されていた **ON KEY** ステートメントなどは Visual Basic for MS-DOS でもサポートされていますが、フォームが表示されているときに実行しようとするとエラーになります。

参照　KeyPress イベント

KeyPress イベント

機能　ユーザーがキーボードから文字を入力したときに、そのときフォーカスを持っているオブジェクトに対して発生します。

構文

SUB Form_KeyPress (*KeyAscii* **AS INTEGER)**

SUB *ctlname***_KeyPress ([***Index* **AS INTEGER]** *KeyAscii* **AS INTEGER)**

KeyAscii　入力された文字に対応する文字コード (ASCII コードまたは漢字連続コード) が渡されます。

Index　コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，水平スクロール バー，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

KeyPress イベントは、ユーザーが押したキーに従って処理を記述するときに便利です。キーボードから文字が入力されると、そのときフォーカスを持っているオブジェクトに対してイベントが発生します。フォームは、そのフォーム上に有効なコントロールが表示されていないときだけ、このイベントを認識します。

KeyPress イベントは、次に示す文字がキーボードから入力されたときに発生します。

- 出力可能なすべてのキーボード上の文字 (アルファベット，数字，記号，スペースなど)
- [CTRL] キーとアルファベットまたはいくつかの特殊キーのひとつとの組み合わせ
- [ENTER] キー
- [BS] キー
- 日本語入力システムを通じて入力された文字 (テキスト ボックスとコンボ ボックスのみ)

大文字小文字は区別して扱われます。

KeyPress イベントは、テキスト ボックスやコンボ ボックスに対するユーザーの入力をチェックするときに便利です。入力された文字を 1 つずつチェックしたり、入力と同時に書式付けするときなどに使います。

KeyAscii 引数には、入力された文字に対応する文字コード (ASCII コードまたは漢字連続コード) が参照によって引き渡されます。この値を変更すれば、オブジェクトに入力されたものとは別の文字を送ることができます。この機能は、入力されたパスワードを画面に表示しないようにするときなどに便利です。

KeyAscii 引数の値を 0 に変更すると、オブジェクトは文字を受け取りません。これによって、キー入力をキャンセルすることができます。

KeyAscii 引数に渡される値は文字コードであり、**CHR$** 関数で文字に変換できます。

テキスト ボックスやコンボ ボックスにフォーカスがあるとき、日本語入力システムをオンにすることができます。日本語入力システムを通じて文字を入力する場合、KeyPress イベントは、変換が確定したときに、1 文字につき 1 回発生します。

ファンクション キーや [DEL] キーなど、KeyPress イベントでは扱えないキーの入力を検出するには、KeyDown または KeyUp イベントを使います。

参照

Change イベント，KeyDown イベント，KeyUp イベント

KILL ステートメント

機能　ディスク ファイルを削除します。

構文　**KILL** *filespec$*

filespec$	削除するファイルを示す文字列式。ドライブ名やディレクトリ パス，MS-DOS のワイルドカード (? と *) を含めることができます。

解説　**KILL** ステートメントは、MS-DOS の DEL や ERASE コマンドと同じ働きをします。

KILL ステートメントは、プログラム ファイル，ランダムアクセス ファイル，シーケンシャル ファイルなど、どんな種類のディスク ファイルにも使えます。*filespec$* には、? または * のワイルドカード文字も使えます。"?" は任意の 1 文字に対応し、"*" は、その位置から始まる任意の長さの任意の文字列に対応します。

KILL ステートメントを使って削除できるのは、ファイルだけです。ディレクトリを削除するには、**RMDIR** ステートメントを使ってください。

現在オープンしているファイルを削除しようとすると、"すでにオープンされています" エラーが発生します。

参照　**FILES** ステートメント

KINSTR 関数

機能　ある文字列の中から別の文字列を検索し、最初に見つかった位置を返します。

構文　**KINSTR**([*start%*,]*string-expression1$*, *string-expression2$*)

start%	*string-expression1$* の中で、検索を開始する位置を先頭からの文字数で示す、1 ～ 32,767 の整数。省略すると、*string-expression1$* の先頭と見なされます。
string-expression1$	検索する文字列。
string-expression2$	検索対象となる文字列。

解説　**KINSTR** 関数は、*string-expression1$* に指定した文字列の中から、*string-expression2$* に指定した文字列に一致する部分を探します。この関数の機能は **INSTR** 関数と同じですが、文字列中に 2 バイト文字が含まれていたときの扱いが異なります。**INSTR** 関数がバイト位置を返すのに対して、**KINSTR** 関数は文字位置を返します。同様に検索を開始する位置 *start%* も、文字位置で指定します。

参照　**INSTR** 関数，**KLEN** 関数

KLEN 関数

機能　指定した文字列の文字数を返します。

構文　**KLEN**(*string-expression$*)

string-expression$　文字数を調べる文字列。

解説　この関数の機能は **LEN** 関数と同じですが、文字列中に 2 バイト文字が含まれていたときの扱いが異なります。**LEN** 関数がバイト数を返すのに対して、**KLEN** 関数は文字数を返します。したがって **LEN**(*string-expression$*)-**KLEN**(*string-expression$*) は、文字列中の 2 バイト文字の個数を示します。

参照　**LEN** 関数

KMID$ 関数

機能　文字列の中から、指定した文字数分の文字列を返します。

構文　**KMID$**(*string-expression$*, *start%*[, *length%*])

string-expression$　元となる文字列。

start%　文字列を取り出す開始位置を先頭からの文字数で示す、 1～32,767 の整数。

length%　取り出す文字列の文字数を示す、 1～32,767 の整数。省略すると、*start%* が示す位置以降の、元となる文字列の最後までと見なされます。

解説　**KMID$** 関数は、*string-expression$* で指定した文字列の、先頭から *start%* 文字目から *length%* 文字分を取り出して返します。

この関数の機能は **MID$** 関数と同じですが、文字列中に 2 バイト文字が含まれていたときの扱いが異なります。**MID$** 関数がバイト数で指定するのに対し、**KMID$** 関数は文字数で指定します。

string-expression$ の文字数を調べるときは、**KLEN** 関数を使ってください。

参照　**KLEN** 関数，**MID$** 関数，**KMID$** ステートメント

KMID$ ステートメント

機能

文字列変数の一部を別の文字列に置き換えます。

構文

**KMID$**(*stringvariable$*, *start%*[, *length%*]) = *string-expression$*

stringvariable$	変更する文字列を含む文字列変数。
start%	文字列の置き換えを始める位置を、*stringvariable$* の先頭からの文字数を示す整数。
length%	置き換える文字列の文字数を示す整数。省略すると、*start%* が示す位置以降の、*string-expression$* 全体の文字数分が置き換えられます。
string-expression$	新しく置き換える文字列。

解説

**KMID$** ステートメントは、*stringvariable$* で指定した文字列の、*start%* 文字目から *length%* 文字分を、*string-expression$* の先頭から *length%* 文字分で置き換えます。

start% が示す位置以降の文字数が *stirng-expression$* の文字数より短ければ、余りの部分は置き換えられません。*length%* の指定にかかわらず、置き換え後に *stringvariable$* の文字数が増えることはありません。

このステートメントの機能は **MID$** ステートメントと同じですが、文字列中に 2 バイト文字が含まれていたときの扱いが異なります。**MID$** ステートメントがバイト数で指定するのに対し、**KMID$** ステートメントは文字数で指定します。

このステートメントを使うと、2 バイト文字を 1 バイト文字に、1 バイト文字を 2 バイト文字に置き換えることができます。この場合、*stringvariable$* の文字数は変わりませんが、バイト数は変わります。

参照

KMID$** 関数，**MID$ ステートメント

KPOS 関数

機能　文字列中の指定した文字位置までのバイト数を返します。

構文　**KPOS**(*stringvariable$*, *characternumber%*)

stringvariable$　バイト数を調べたい文字列。
characternumber%　先頭からの文字数を示す整数。

解説　**KPOS** 関数は、*stringvariable$* に指定した文字列の *characternumber%* 文字目の文字の、先頭からのバイト位置を返します。*characternumber%* が *string-expression$* の長さより長いと、0 を返します。

参照　**KLEN** 関数

KTN$ 関数

機能　指定した文字列の最初の文字の区点コードを文字列として返します。

構文　**KTN$**(*string-expression$*)

string-expression$　調べたい文字列。

解説　*string-expression$* に指定した文字列の、先頭 1 文字が 2 バイト文字なら、その文字の区点コードを 10 進数で表した 4 バイトの文字列を返します。先頭の 1 文字が 1 バイト文字なら、その文字の ASCII コードを 10 進数で表した 3 バイト以下の文字列を返します。

参照　**ASC** 関数，**JIS$** 関数

LargeChange, SmallChange プロパティ

機能

ユーザーがスクロール バー上のボックスと矢印の間の領域をクリックしたときにスクロールする量 (LargeChange)、およびスクロール バーの矢印をクリックしたときに変化する量 (SmallChange) を決めます。

構文

[*formname.*]*ctlname*.**LargeChange**[= *change%*]

[*formname.*]*ctlname*.**SmallChange**[= *change%*]

ctlname	水平スクロール バーまたは垂直スクロール バーの名前。
change%	Value プロパティの値が変化する量を示す、1～32767 の整数。デフォルトは 1 です。

対象

水平スクロール バー，垂直スクロール バー

解説

水平スクロール バー コントロールおよび垂直スクロール バー コントロールでは、ユーザーがスクロール バー上のボックスと矢印の間の領域や矢印自身をクリックしたときに、どの程度スクロールするかをあらかじめ設定して置く必要があります。LargeChange プロパティはボックスと矢印の間の領域がクリックされたときの変化量を示し、SmallChange プロパティは上下の矢印がクリックされたときの変化量を示します。

スクロール バーのボックスがどの位置にあるかは、Value プロパティが示しています。つまり、ユーザーがスクロール バー上をクリックしたとき、LargeChange および SmallChange プロパティで設定されている値だけ、Value プロパティの値が変化します。

スクロール バーの設定可能範囲、つまり Value プロパティの値の最小値と最大値を設定するには、Max および Min プロパティを使います。

通常、LargeChange および SmallChange プロパティの値はデザイン時に設定しますが、必要に応じてコード中から変更することもできます。

リスト ボックス，コンボ ボックス，テキスト ボックスのスクロール バーでは、ユーザーの操作によってどのくらいスクロールするかは、表示しているデータの量に比例して自動的に設定されます。

参照

Change イベント，Max プロパティ，Min プロパティ，Value プロパティ

LBOUND, UBOUND 関数

機能

配列の指定した次元の添字の最小値 (**LBOUND**) または最大値 (**UBOUND**) を返します。

構文

LBOUND(*array*[, *dimension%*])

UBOUND(*array*[, *dimension%*])

array	配列変数の名前。
dimension%	添字の最小値または最大値を調べたい配列の次元を示す整数。最初の次元なら 1, 2 番目の次元なら 2 というように指定します。配列が 1 次元の場合は省略できます。

解説

次のような配列が宣言されているとき、**LBOUND** 関数は、下の表のような値を返します。

```
DIM A(1 TO 100, 0 TO 50, -3 TO 4)
```

関数	戻り値	関数	戻り値
LBOUND(A,1)	1	UBOUND(A,1)	100
LBOUND(A,2)	0	UBOUND(A,2)	50
LBOUND(A,3)	-3	UBOUND(A,3)	4

REDIM ステートメントを使うと、動的配列の各次元の添字の最小値と最大値を変更できます。

DIM または **REDIM** ステートメントで配列を宣言するとき、添字の最小値の指定を省略すると、最小値はデフォルトの 0 または 1 になります。デフォルトが 1 になるのは、**OPTION BASE** ステートメントで 1 に設定されているときだけです。

参照

DIM ステートメント，**REDIM** ステートメント，**OPTION BASE** ステートメント

LCASE$, UCASE$ 関数

機能

LCASE$** 関数は、指定した文字列中の、アルファベットの大文字を小文字に変換する文字列処理関数です。**UCASE$ 関数は、指定した文字列中の、アルファベットの小文字を大文字に変換する文字列処理関数です。

構文

**LCASE$**(*string-expression$*)

**UCASE$**(*string-expression$*)

string-expression$　置き換える文字列。文字列定数，文字列変数，文字列式のいずれかでも指定できます。また、固定長でも可変長でもかまいません。

解説

LCASE$** 関数と **UCASE$ 関数は、大文字と小文字を区別せずに文字列を比較するときに使ってください。

Left, Top プロパティ

機能

フォームまたはコントロールの水平位置または列座標 (Left)、垂直位置または行座標 (Top) を決めます。

構文

[*formname.*][*ctlname.*]**Left**[= *horipos%*]

[*formname.*][*ctlname.*]**Top**[= *vertpos%*]

horipos%　コンテナの左上角からの水平方向の相対位置を示す、 0～254 の整数 (列座標)。

vertpos%　コンテナの左上角からの垂直方向の相対位置を示す、 0～254 の整数 (行座標)。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，タイマー，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

Left と Top プロパティの値は、常にキャラクタ単位で表されます。

Left プロパティは、コンテナまたは画面の左上角を原点 (0, 0) とした、そのフォームまたはコントロールの左上角の列座標を示し、Top プロパティは行座標を表します。フォームの Left および Top プロパティは、画面左上角に対する相対位置です。ただし MDI アプリケーションの子フォームは、親フォームに対する相対位置になります。コントロールの Left および Top プロパティは、フォームの左上角に対する相対位置で

す。ただしピクチャー ボックスやフレームの子コントロールは、コンテナ (ピクチャー ボックスまたはフレーム) に対する相対位置になります。

ユーザーやコード中からの操作によってオブジェクトが移動されると、これらのプロパティの値が変化します。

Left および Top プロパティや、関連するプロパティである Height および Width プロパティは、オブジェクトのサイズ変更や移動などの、オブジェクト自身の位置や領域の計算が必要なときに使います。これに対して ScaleHeight および ScaleWidth プロパティは、文字の出力や子コントロールの移動など、オブジェクト内部の領域の計算が必要なときに使います。

タイマーコントロールは実行時には表示されないため、これらのプロパティは実行時には無意味です。

参照　**LEFT$** 関数，Height プロパティ，**MOVE** メソッド，Width プロパティ

LEFT$, RIGHT$ 関数

機能　**LEFT$** 関数は、文字列の左端から指定したバイト数分の文字列を取り出す文字列処理関数です。**RIGHT$** 関数は、文字列の右端から指定したバイト数分の文字列を取り出す文字列処理関数です。

構文

**LEFT$**(*string-expression$, n%*)

**RIGHT$**(*string-expression$, n%*)

string-expression$	元となる文字列。
n%	取り出す文字列のバイト数を示す、1～32,767 の整数。0 を指定すると、null 文字列が返されます。また *string-expression$* の文字数より多い値を指定すると、*string-expression$* 全体が返されます。

解説　文字列のバイト数を調べるには **LEN** 関数を、文字数を調べるには **KLEN** 関数を使ってください。

LEN 関数

機能　指定した文字列のバイト数、または変数に必要なバイト数を返す文字列処理関数です。

構文　**LEN**({*string-expression$*|*variable*})

string-expression$	バイト数を調べる文字列。
variable	必要なバイト数を調べる変数。

解説　引数に文字列変数を指定すると、その変数が必要とするバイト数を返します。ランダム アクセス ファイルのレコードの正確なサイズを得るときに便利です。

文字列式を指定すると、その文字列の長さをバイト数で返します。文字数を調べるには **KLEN** 関数を使ってください。

LET ステートメント

機能　式の値を変数に代入します。

構文　[**LET**]*variable* = *expression*

variable	代入する変数。
expression	変数に代入したい値を持つ式を指定します。

解説　**LET** キーワードは、つねに省略可能です。ステートメント中の等号 (=) は、そのステートメントが代入ステートメントであることを表します。

LET ステートメントを使って、ユーザー定義型の変数同士の代入もできます。ただし両方の変数は、同じユーザー定義型でなければなりません。異なるユーザー定義型の変数を代入するには、**LSET** ステートメントを使ってください。

参照　**LSET** ステートメント

LINE ステートメント

機能

直線や四角形を描きます。フォームが表示されている状態では利用できません。

構文

LINE[[**STEP**](*x1!,y1!*)] - [**STEP**](*x2!,y2!*)[, [*color%*][, [**B**|**BF**][, *style%*]]]

STEP	座標を相対座標で指定することを示します。
(*x1!,y1!*)	直線の始点、または四角形の頂点のビューポート座標。
(*x2!,y2!*)	直線の終点、または四角形の頂点のビューポート座標。
color%	直線や四角形の色を示すパレット番号。省略すると、グラフィックスのデフォルトの色と見なされます。
B	直線ではなく、指定した座標 (*x1!,y1!*) と (*x2!,y2!*) を対角とする四角形を描くことを示します。
BF	四角形の内部を塗りつぶすことを示します。
style%	直線または四角形の境界線のラインスタイルを示す、 16 ビットの 10 進数または 16 進数。

解説

フォームが表示されているときに **LINE** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **LINE** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

ビューポート座標は、**WINDOW** ステートメントが実行されていなければ物理座標と同じです。物理座標は、**VIEW** ステートメントが実行されていなければ、画面左上角を原点とする絶対座標です。

座標 (*x1!,y1!*) と (*x2!,y2!*) には、直線の両端の座標を指定します。2 つの座標を指定する順序は、重要ではありません。座標 (10,20) から座標 (120,130) に引いた直線は、座標 (120,130) から座標 (10,20) に引いた直線と同じです。

STEP キーワードは、現在のグラフィックス出力位置 (直前にプロットされた点) からの相対座標であることを示します。例えば、**STEP** キーワードを使うと次のように指定できます。現在のグラフィックス出力位置は、(10,10) であるとします。

ステートメント	引かれる直線の両端の座標
LINE -(50,50)	(10,10) から (50,50)
LINE -STEP(50,50)	(10,10) から (60,60)
LINE(25,25)-STEP(50,50)	(25,25) から (75,75)
LINE STEP(25,25)-STEP(50,50)	(35,35) から (85,85)
LINE STEP(25,25)-(50,50)	(35,35) から (50,50)

CLS または **SCREEN** ステートメントを使って画面を初期化すると、グラフィックス出力位置も初期化されます。また、**PSET, PRESET, CIRCLE, DRAW** ステートメントも、グラフィックス出力位置に影響します。

style% には、直線または四角形の境界線の、ラインスタイル (線種) を指定します。color で指定した色にしたいピクセルを 1、色を付けないピクセルを 0 で表した、16 ビットの 10 進数または 16 進数を指定します。描かれる線は、このパターンの繰り返しで描かれます。1 のビットに対応するピクセルは *color%* の色でプロットされ、0 のビットに対応するピクセルは元の色のままです (以前に何も描かれていなければ背景色)。内部を塗りつぶす四角形を描く場合 (**BF** を指定した場合) は、*style%* を指定できません。

現在のビューポートの外にある点を指定すると、ビューポートの外側の部分は描かれません。

参照　**CIRCLE** ステートメント，**PSET** ステートメント，**COLOR** ステートメント，**SCREEN** ステートメント，**LINE INPUT** ステートメント，**LINE INPUT #** ステートメント

LINE INPUT ステートメント

INPUT ステートメントを参照してください。

LINE INPUT # ステートメント

INPUT # ステートメントを参照してください。

List プロパティ

機能　コントロールのリスト ボックス部分の項目の取得や設定を行います。デザイン時には使用できません。ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックスでは読み込み専用です。

構文　[*formname.*]*ctlname*.**List**(*index%*)[= *string-expression$*]

ctlname	コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，リスト ボックスの名前。
index%	リスト中の項目を示すインデックス番号 (文字列配列の添字)。
string-expression$	その項目に設定するテキストを示す文字列式。

対象　コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，リスト ボックス

解説

リストを表示するコントロールでは、リスト ボックス部分に含まれる各項目は、文字列の配列に格納されています。List プロパティは、この配列の各要素、つまりリスト ボックス部分に表示される各文字列を設定したり取得したりします。配列の添字をインデックス番号と呼びます。

ディレクトリ リスト ボックス以外のコントロールでは、配列の添字 (インデックス番号) は 0 から始まり、添字の最大値は "項目数 -1" です。項目数は、ListCount プロパティで調べることができます。

ディレクトリ リスト ボックスでは、インデックス番号の割り振り方が他のコントロールと異なります。ディレクトリの一覧を表示するときの中心となるディレクトリを、展開されているディレクトリと呼びます。例えば、ユーザーが C:¥BIN¥MYPROG を選択 (反転表示) していたのであれば、C:¥BIN¥MYPROG が展開されているディレクトリです。このディレクトリ中のすべてのサブディレクトリは、リスト ボックス部分に表示されます。これらのサブディレクトリを指すインデックス番号は、0～ListCount-1 の範囲になります。また、そのとき展開されているディレクトリのインデックス番号は -1 です。さらに、そのドライブのルート ディレクトリからそのとき展開されているディレクトリの間のすべてのディレクトリも表示されます。例えば、そのとき展開されているディレクトリが C:¥BIN¥MYPROG なら、"C:¥", "C:¥BIN" も表示されます。このとき、"C:¥" のインデックス番号は -3, "C:¥BIN" は -2 になります。つまり、そのとき展開されているディレクトリまでの階層数を n とすると、ディレクトリ リスト ボックスのインデックス番号は、-n～ListCount-1 の範囲になります。

List プロパティを使い、(範囲外のインデックス番号を指定して) 存在しない項目のテキストを取得しようとしたとき、null 文字列 ("") が返されます。このときエラーにはなりません。

コンボ ボックスとリスト ボックスを生成した直後は、何も項目がない状態です。このとき、ListIndex プロパティの値は -1 です。また、リスト ボックス部分で項目が選択されていないときも、ListIndex プロパティの値は -1 になります。インデックス番号 (*index%* 引数) に -1 を指定したときの List プロパティの値は null 文字列 ("") です。

コンボ ボックスやリスト ボックスに対して項目を追加するには、**ADDITEM** メソッドを使います。通常は、フォームの Load イベントに対するイベント プロシージャ中で項目を設定します。項目をアルファベット順に並べるには、項目を追加する前にコントロールの Sorted プロパティを True (-1) に設定しておきます。項目を削除するには **REMOVEITEM** メソッドを使います。

コンボ ボックスやリスト ボックスの List プロパティを使うと、すでに設定されている項目のテキスト (表示されている文字列) を取得したり、新しい文字列に変更したりできます。ただし、項目を追加することはできません。

ファイル関係のコントロール (ディレクトリ リスト ボックス, ドライブ リスト ボックス, ファイル リスト ボックス) では、項目は自動的に表示されるため、設定する必要はありません。List プロパティは読み込み専用で、設定しようとするとエラーになります。

ファイル関係のコントロールでは、リスト ボックス部分に次のような項目が表示されます。

コントロール	値
ディレクトリ リスト ボックス	そのとき展開されているディレクトリに含まれるサブディレクトリの一覧 (インデックス番号 0 ～ ListCount-1)，そのとき展開されているディレクトリ名 (インデックス番号 -1)，それより上の階層のディレクトリ名 (インデックス番号 -2 ～ - 階層数)
ドライブ リスト ボックス	接続されているドライブの一覧
ファイル リスト ボックス	そのとき展開されているディレクトリに含まれるファイルのうち、Pattern プロパティに適合するファイルの一覧 (サブディレクトリは含まれない)
ファイル リスト ボックス	Pattern プロパティ

参照

ADDITEM メソッド，ListCount プロパティ，ListIndex プロパティ，**REMOVEITEM** メソッド，Sorted プロパティ

ListCount プロパティ

機能

コントロールのリスト ボックス部分の現在の項目数を返します。デザイン時には使用できず、実行時には読み込み専用です。

構文

[*formname*.]*ctlname*.**ListCount**

ctlname　コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，リスト ボックスの名前。

対象

コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，リスト ボックス

解説

ディレクトリ リスト ボックスを除くコントロールでは、ListCount プロパティは、各コントロールのリスト ボックス部分に含まれる項目数を返します。

ディレクトリ リスト ボックスでは、ListCount プロパティは、そのとき展開されているディレクトリに含まれるサブディレクトリ数を返します。そのとき展開されているディレクトリ自身や、より上の階層のディレクトリ数は含まれませんから注意してください。

ファイル関係のコントロールでは、ListCount プロパティの値は次のような数を示します。

コントロール	値
ディレクトリ リスト ボックス	そのとき展開されているディレクトリに含まれるサブディレクトリ数
ドライブ リスト ボックス	接続されているドライブ数
ファイル リスト ボックス	そのとき展開されているディレクトリに含まれるファイルのうち、Pattern プロパティに適合するファイルの数

リスト ボックス部分に項目がないときは、ListCount プロパティの返す値は 0 です。

参照

ADDITEM メソッド，List プロパティ，ListIndex プロパティ，**REMOVEITEM** メソッド

ListIndex プロパティ

機能

コントロールのリスト ボックス部分で、現在選択 (反転表示) されている項目のインデックス番号の取得や設定を行います。デザイン時には使用できません。

構文

[*formname*.]*ctlname*.**ListIndex**[= *index*%]

ctlname	コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，リスト ボックスの名前。
index%	リスト中の選択 (反転表示) されている項目を示すインデックス番号 (文字列配列の添字)。

対象

コンボ ボックス，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，リスト ボックス

解説

リストを表示するコントロールでは、リスト ボックス部分に含まれる各項目は、文字列の配列に格納されています。配列の添字をインデックス番号と呼びます。ListIndex プロパティは、そのとき選択 (反転表示) されている項目のインデックス番号を取得したり、別の項目を選択状態に設定したりするために使います。存在しない項目に設定しようとするとエラーになります。

ディレクトリ リスト ボックス以外のコントロールでは、配列の添字 (インデックス番号) は 0 から始まり、添字の最大値は "項目数 -1" です。項目数は、ListCount プロパティで調べることができます。

ディレクトリ リスト ボックスでは、インデックス番号の割り振り方が他のコントロールと異なります。ディレクトリの一覧を表示するときの中心となるディレクトリを、展開されているディレクトリと呼びます。例えば、ユーザーが C:¥BIN¥MYPROG を選択 (反転表示) していたのであれば、C:¥BIN¥MYPROG が展開されているディレクトリです。このディレクトリ中のすべてのサブディレクトリは、リスト ボックス部分に表示されます。これらのサブディレクトリを指すインデックス番号は、0～ListCount-1 の

範囲になります。また、そのとき展開されているディレクトリのインデックス番号は -1 です。さらに、そのドライブのルート ディレクトリからそのとき展開されているディレクトリの間のすべてのディレクトリも表示されます。例えば、そのとき展開されているディレクトリが C:¥BIN¥MYPROG なら、"C:¥", "C:¥BIN" も表示されます。このとき、"C:¥" のインデックス番号は -3, "C:¥BIN" は -2 になります。つまり、そのとき展開されているディレクトリまでの階層数を n とすると、ディレクトリ リスト ボックスのインデックス番号は、-n～ListCount-1 の範囲になります。

リスト ボックス部分でそのとき選択 (反転表示) されている項目の文字列を取得するには、List プロパティと ListIndex プロパティの組み合わせか Text プロパティを使います。次の 2 つは同じ意味です。

```
ctlname.Text
ctlname.List(ctlname.ListIndex)
```

別の項目を選択するには、ListIndex プロパティの値を変更します。このとき、Click イベントが発生します。

ListIndex プロパティの値は、次の条件のとき -1 になります。

- ディレクトリ リスト ボックス以外のすべてのコントロールで、項目が選択されていないとき。または項目が 1 つもないとき。
- コンボ ボックスで、リスト ボックス部分の項目の選択ではなく、テキスト ボックス部分へ直接入力が行われているとき (リスト ボックス部分に反転表示されている項目はない)。
- ディレクトリ リスト ボックスで、そのとき展開されているディレクトリが選択されているとき。

ListIndex プロパティの初期値は、コントロールの種類によって次のようになっています。

コントロール	値	意味
コンボ ボックス	-1	項目が選択されていない
ディレクトリ リスト ボックス	-1	コントロールがロードされた時点でのカレント ディレクトリ
ドライブ リスト ボックス	n	コントロールがロードされた時点でのカレント ドライブのインデックス番号
ファイル リスト ボックス	-1	項目が選択されていない
リスト ボックス	-1	項目が選択されていない

参照

ADDITEM メソッド，List プロパティ，ListCount プロパティ，**REMOVEITEM** メソッド

Load イベント

機能　フォームがメモリにロードされるときに発生します。

構文　**SUB Form_Load()**

対象　フォーム

解説　Load イベントは、フォームの初期化を行うときに便利です。例えば、各コントロールのデフォルトの設定を指定したり、そのフォーム全体で有効な変数を初期化したりできます。

Load イベントは、フォームが最初にロードされたときに発生します。フォームのロード方法には、暗黙のロードと明示的なロードの 2 種類があります。ロードされていないフォームが参照されると、そのフォームは自動的にロードされます。これを暗黙のロードと言います。また、フォームを明示的にロードするには、**LOAD** ステートメントを使います。

実行開始フォームは、Visible プロパティが True (-1) に設定されていれば、自動的に画面に表示されます。それ以外のフォームは、Visible プロパティが True (-1) に設定されていても、ロード時に自動的には表示されません。ロードされているフォームを表示するには、**SHOW** メソッドを実行するか、Visible プロパティをもう一度 True (-1) に設定する必要があります。

Load イベントが発生した後、自動的に Resize, GotFocus, Paint イベントが、この順番で発生します。ただし Paint イベントが発生するのは、AutoRedraw プロパティが False (0) に設定されているときだけです。

参照　**LOAD** ステートメント，UnLoad イベント，**UNLOAD** ステートメント

LOAD ステートメント

機能　フォームをメモリにロードしたり、コントロール配列にコントロールを加えたりします。

構文　**LOAD** {*formname*|*ctlname*[*index%*]}

formname	生成またはロードするフォームの名前。
ctlname	要素が加えられるコントロール配列。
index%	コントロール配列中の位置を表す整数。

解説

フォームに対しては、明示的に **LOAD** ステートメントを実行する必要はありません。フォームは、参照されたときにロードされていなければ、自動的にロードされます。例えば **SHOW** メソッドは、フォームがロードされていない場合はロードしてから表示します。

アプリケーションの実行開始ファイルがフォーム モジュールで、デザイン時にそのフォームの Visible プロパティを True (-1) に設定してあれば、そのフォームは実行開始時に自動的にロードされて表示されます。

アンロードされた、またはまだロードされていないフォームに対して **HIDE** メソッドを使うと、そのフォームを非表示のままロードすることができます。

ロードされているが表示されていないフォームを表示するには、**SHOW** メソッドを使います。

アンロードされた、またはまだロードされていないフォームや、そのフォーム上のコントロールへの参照を行うと、そのフォームは再ロードされます。つまり、フォームがロードされていない状態でも、そのフォームに関連するプロパティの値を変更しようとすると、フォームは自動的にロードされます。ただしプロパティによっては、デザイン時にしか操作できないものがありますから注意してください。

すでにロードされているフォームをロードしようとしても無視されます。

フォームがロードされるとき、すべてのプロパティが初期値に設定されてから、Load イベントに対するイベント プロシージャが実行されます。

Visual Basic では、1 つのフォーム モジュールから複数のインスタンスを生成することはできません。つまり、1 つのフォーム モジュールで表示できるフォームは 1 度に 1 つです。ただし、デザイン時にコントロール配列として定義されたコントロールは、複数のインスタンスをロードおよび表示できます。例えばデザイン時に、チェック ボックスを 1 つしか配置しない場合でも、それをチェック ボックスの配列の要素として定義しておきます。これによってアプリケーションは、実行時に特定の条件に応答して新しいチェック ボックスを配列にロードし、発生するイベントを認識することができます。

MSGBOX または **INPUTBOX$** 関数で表示されるダイアログ ボックスを明示的にロード，アンロードしたり、**SHOW** メソッドを使って表示することはできません。

参照

Load イベント，**SHOW** メソッド，**UNLOAD** ステートメント

LOC 関数

機能　指定したファイル内で次に読み書きする位置を返します。

構文　**LOC**(*filenumber%*)

filenumber%　オープンしているファイルまたはデバイスのファイル番号。

解説　**LOC** 関数は、SCRN, KYBD, LPT*n* デバイスには使用できません。

COM デバイスに対して **LOC** 関数を実行すると、入力バッファにあるデータのバイト数を返します。戻り値は、デバイスが ASCII モードでオープンされているか、バイナリモードでオープンされているかによって異なります。ASCII モードでは、ファイルの終わり (EOF コード) を受け取ると、すぐにバッファへのデータの読み込みを中止します。EOF コードそのものはバッファに登録されず、読み込まれません。EOF コードを読み込もうとすると、エラーになります。バイナリモードでは EOF コードは無視され、ファイル全体が読み込まれます。最後に読み書きされたバイトが返されます。

ランダムアクセス ファイルに対して **LOC** 関数を実行すると、直前に入出力を行ったレコードの番号を返します。

シーケンシャル ファイルに対して **LOC** 関数を実行すると、ファイル内の現在のバイト位置を、128 で割った値を返します。

PIPE デバイスに対しては無効です。

LOCATE ステートメント

POS 関数 を参照してください。

LOCK, UNLOCK ステートメント

機能

オープンしているファイルの一部または全部に対する、他のプロセスからのアクセスを制御します。**LOCK** ステートメントは指定された範囲を他からアクセスできないようにロックし、**UNLOCK** ステートメントは、そのロックを解除します。

構文

LOCK [#]*filenumber%*[, {*record&*|[*start&*]**TO** *end&*}]

.

.

.

UNLOCK [#]*filenumber%*[, {*record&*|[*start&*]**TO** *end&*}]

filenumber%	オープンしているファイルのファイル番号。
record&	ロックまたはロックを解除するレコードの番号か、バイト位置を指定する 1～2,147,483,647 (=2^{31}-1) の整数。省略すると、ファイル全体が対象となります。1 つのレコードは、最大 32,767 バイトの長さです。
start&	ロックまたはロックを解除する最初のレコードの番号か、バイト位置。省略すると、ファイルの先頭と見なされます。
end&	ロックまたはロックを解除する最後のレコードの番号か、バイト位置。

解説

LOCK ステートメントと **UNLOCK** ステートメントは、複数のプロセスが同じファイルにアクセスする可能性のある、ネットワーク環境で使われます。

LOCK ステートメントと **UNLOCK** ステートメントは必ず 2 つ 1 組で使うので、両方のステートメントに対する引数は一致していなければなりません。

ファイルをクローズしたりプログラムを終了する前に、**UNLOCK** ステートメントを使って、必ずすべてのロックを解除してください。ロックを解除しないと、予測できないエラーの原因になります。

デバイスに対して、**LOCK** および **UNLOCK** ステートメントを実行することはできません。

1 つのレコードだけを指定すると、そのレコードだけがロックまたはアンロックされます。レコードの範囲を指定するときに *start&* を省略すると、先頭のレコードから *end&* までが対象になります。引数をすべて省略すると、ファイル全体が対象になります。

バイナリモード ファイルでは、引数 *record&, start&, end&* は、ファイルの先頭からの相対的なバイト位置を表します。ファイルの最初のバイトは、バイト位置 1 です。

ランダムアクセス ファイルでは、引数 *record&, start&, end&* は、ファイルの先頭からの相対的なレコード番号を表します。ファイルの最初のレコードは、レコード番号 1 です。

シーケンシャルアクセス ファイルでは、*start&* および *end&* の指定にかかわらず、ファイル全体が対象になります。

ロックされているファイルにアクセスしようとすると、"書き込みできません" エラーが発生します。

LOCK および **UNLOCK** ステートメントは、ネットワークをサポートしている MS-DOS バージョン 3.1 以降でのみ有効です。あらかじめ SHARE.EXE を実行しておかなければなりません。バージョン 3.1 より前の MS-DOS 上で **LOCK** または **UNLOCK** ステートメントを実行すると、"この機能はサポートされていません" エラーが発生します。

LOF 関数

機能

オープンしているファイルのサイズ (バイト数)、または出力バッファに残っている空き領域のバイト数を返します。

構文

LOF(*filenumber%*)

filenumber% オープンしているファイルのファイル番号。

解説

対象が通常のファイルの場合は、そのファイルのバイト数を返します。**OPEN COM** ステートメントでファイルとしてオープンしたデバイスの場合には、出力バッファの空きバイト数を返します。

LOF 関数は、ディスク ファイルと COM*n*: デバイスにのみ使えます。他の Basic のデバイス SCRN:, KYBD:, CONS:, LPT*n*: には使えません。

参照

EOF 関数，**LEN** 関数，**LOC** 関数，**OPEN** ステートメント (ファイル I/O)

LOG 関数

EXP 関数を参照してください

LostFocus イベント

機能

オブジェクトがフォーカスを失ったときに発生します。

構文

SUB Form_LostFocus ()

SUB *ctlname*_**LostFocus** ([*Index* **AS INTEGER**])

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，水平スクロール バー，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

フォーカスは、クリックや [TAB] キーを押すなどのユーザーの操作が行われたときや、コード中で **SETFOCUS** メソッドが実行されたときに移動します。

フォームは、コントロールが配置されていないか、配置されているすべてのコントロールが無効または非表示に設定されているときだけ、フォーカスを持つことができます。

Enabled および Visible プロパティが True (-1) に設定されているオブジェクトだけが、フォーカスを得ることがでます。

LostFocus イベントは、更新された内容が有効かどうかを確認するときに便利です。ユーザーがそのコントロールに対する設定を終えたときに、内容を確認するようにします。LostFocus イベントは、あるコントロールがフォーカスを失ったときに、その他のコントロールを無効または有効にするために利用できます。また、GotFocus イベントのイベント プロシージャで行った変更を、前の状態に戻すためにも便利です。

コントロール間のフォーカスの移動をキーボードから操作できるようにするには、タブ オーダーを設定するか、フォーム上のコントロールにアクセス キーを割り当てます。

コントロールがフォーカスを得たり失ったりするたびに、コントロールの色を変更すると、視覚的に分かりやすいユーザー インターフェイスになります。

参照

ActiveControl プロパティ，ActiveForm プロパティ，GotFocus イベント，**SETFOCUS** メソッド，TabIndex プロパティ，TabStop プロパティ

LPOS 関数

機能　プリンタ バッファ内のプリンタ ヘッドの現在位置を返します。

解説　**LPOS**(*n*%)

n%　対象となるプリンタ ポートの番号を表す 0〜3 の整数。0 または 1 が LPT1:, 2 が LPT2:, 3 が LPT3: を示します。

解説　**LPOS** 関数は、タブ文字をスペースに拡張しないので、必ずしもプリンタ ヘッドの物理的な位置を返すわけではありません。また、プリンタが独自のバッファを持っていることもあり、この場合もプリンタ ヘッドの物理的な位置を返しません。

LPRINT ステートメント

機能　数値または文字列データをプリンタに出力します。指定した書式に従って出力することもできます。フォームが表示されている状態では利用できません。

構文　**LPRINT** [*expression*[{,|;}*expression*]...[{,|;}]]

LPRINT USING *formatstr$*; [*expression*[; *expression*]...[;]]

expression　出力するデータを示す数式または文字列式。省略すると、単に改行されます。複数の式を指定するときは、セミコロン (;) またはカンマ (,) で区切ります。

;　データを出力したあと、出力位置を進めないことを示します。次のデータは、この直後から出力されます。 **USING** キーワードを指定した場合、*expression* の間のセミコロンは単なる区切りで、最後のセミコロンだけが意味を持ちます。

,　データを出力したあと、次の出力領域 (14 桁ごと) に出力位置を進めることを示します。次のデータは、次の出力領域の先頭から出力されます。

formatstring$　出力するデータの書式を示す文字列式。

解説　**LPRINT** ステートメントは、出力先が画面ではなくシーケンシャル ファイルであることを除き、**PRINT** ステートメントと同じ機能です。また **USING** キーワードを使って書式を指定して出力するときも、**PRINT** ステートメントで **USING** キーワードを指定したときと同じです。

LPRINT ステートメントはプリンタ デバイス LPT1 を使います。このため、OPEN "LPT1" ステートメントのあるプログラム中では **LPRINT** ステートメントを使えません。実行すると、結果は予測できません。

LPRINT ステートメントは、プリンタの 1 行が半角 80 文字の幅であると想定しています。この幅は、**WIDTH** ステートメントを使って変更することができます.

LPRINT CHR$(13) を実行すると、LPRINT CHR$(13)+CHR$(10) であると見なされます。また、出力する行の長さがプリンタの 1 行の桁数を超える場合、次の行に折り返して出力が続けられます。

参照　**PRINT** ステートメント, **PRINT** メソッド, **PRINT #** ステートメント, **SPC** 関数, **TAB** 関数

LSET, RSET ステートメント

機能　**PUT** ステートメントを使ってデータを書き込むとき、データをランダムアクセス ファイル バッファに格納します。**LSET** ステートメントは文字列変数に左詰めで格納し、**RSET** ステートメントは右詰めで格納します。**LSET** ステートメントは、ユーザー定義型の変数を別のユーザー定義型の変数にコピーするためにも使えます。

構文

LSET *stringvariable$ = string-expression$*

LSET *recordvariable1 = recordvariable2*

RSET *stringvariable$ = string-expression$*

stringvariable$	**FIELD** ステートメントで定義されたランダムアクセス ファイルのフィールド、または文字列変数。
string-expression$	左または右詰めで *stringvariable$* に格納される値。
recordvariable1	コピー元のユーザー定義型 (レコード) 変数。
recordvariable2	コピー先のユーザー定義型 (レコード) 変数。

解説　*string-expression$* の長さが **FIELD** ステートメントで定義した *stringvariable$* のフィールド長よりも短いと、**RSET** ステートメントはフィールド内で文字列を右詰めに、**LSET** ステートメントは左詰めにして、余った部分をスペースで埋めます。文字列がフィールド長よりも長いと、余った部分は切り捨てられます。数値は、あらかじめ **STR$** 関数で文字列に変換してください。

FIELD ステートメントで定義していない文字列変数と、**LSET** または **RSET** ステートメントを使って、指定したフィールドに文字列を左詰めまたは右詰めにすることができます。次の例では、文字列 N$ を半角 20 文字のフィールドに右詰めで代入しています。

```
A$ = SPACE$(20)
RSET A$=N$
```

LSET ステートメントは、あるユーザー定義型の (レコード) 変数を、別のユーザー定義型変数に代入するためにも使うことができます。次の例では、ユーザー定義型変数 RecTwo の内容を別の変数 RecOne にコピーしています。

```
TYPE TwoString
    StrFld AS STRING * 2
END TYPE
TYPE ThreeString
    StrFld AS STRING * 3
END TYPE
DIM RecOne AS TwoString, RecTwo AS ThreeString
.
.
.
LSET RecOne = RecTwo
```

この例では、別に定義されたユーザー定義型の変数の間で代入していることに注意してください。TwoString 型の変数 RecOne のレコード長が 2 バイトなので、ThreeString 型の変数 RecTwo の先頭の 2 バイト分だけがコピーされます。**LSET** ステートメントは、2 つのユーザー定義型変数のうち短いもののバイト数だけをコピーします。同じ型のユーザー定義型変数をコピーするには、**LSET** ステートメントを使ってもかまいませんし、**LET** ステートメントを使うこともできます。

RSET ステートメントに可変長文字列変数を指定すると、文字列は固定長フィールドとして扱われます。**RSET** ステートメントを実行すると、フィールドの長さは、*stringvariable$* の持つ値と同じ長さになります。

参照　**FIELD** ステートメント，**PUT** ステートメント (ファイル I/O)

LTRIM$, RTRIM$ 関数

機能　**LTRIM$** 関数は、指定した文字列から先頭のスペースを削除した文字列を返す文字列処理関数です。**RTRIM$** 関数は、指定した文字列から末尾のスペースを削除した文字列を返す文字列処理関数です。

構文　**LTRIM$**(*string-expression$*)

**RTRIM$**(*string-expression$*)

string-expression$　元になる文字列。

解説　**LTRIM$** と **RTRIM$** 関数はどちらも、固定長文字列変数と可変長文字列変数の両方に対して使用できます。1 バイト，2 バイトを問わず、文字列の先頭または末尾にある空白を削除します。

Max, Min プロパティ

機能

スクロール バーの位置が示す値の最大値 (Max) および最小値 (Min) を決めます。

構文

[*formname.*]*ctlname*.**Max**[= *limit%*]

[*formname.*]*ctlname*.**Min**[= *limit%*]

ctlname	水平スクロール バーまたは垂直スクロール バーの名前。
limit%	Value プロパティの値の最大値または最小値を示す、 -32768～32767 の整数。Max プロパティのデフォルトは 32767，Min プロパティのデフォルトは 0 です。

対象

水平スクロール バー，垂直スクロール バー

解説

水平スクロール バー コントロールおよび垂直スクロール バー コントロールでは、スクロール バー上でのそのときのスクロール ボックスの位置を、Value プロパティの値によって示します。Max プロパティと Min プロパティは、この Value プロパティの最大値と最小値を示します。

水平スクロール バーでは右方向に向かって値が増え、垂直スクロール バーでは下に向かって値が増えます。

Max プロパティに Min プロパティよりも小さい値を指定すると、その設定は無視され Max は Min と同じ値に設定されます。このときスクロール ボックスは、水平スクロール バーなら左端に、垂直スクロール バーなら上端に表示されます。

スクロール バーの移動単位、つまりスクロール バー上のボックスと矢印の間の領域や矢印自身をクリックしたときのスクロール量を設定するには、LargeChange および SmallChange プロパティを使います。

通常、Max および Min プロパティの値はデザイン時に設定しますが、必要に応じてコード中から変更することもできます。

リスト ボックス，コンボ ボックス，テキスト ボックスのスクロール バーでは、スクロールの範囲は、表示しているデータの量に比例して自動的に設定されます。したがってこれらのコントロールには Max および Min プロパティはありません。

参照

LargeChange プロパティ，SmallChange プロパティ，Value プロパティ

MaxButton プロパティ

機能

フォームの右上隅に最大化ボタンを表示するかどうかを決めます。実行時には読み込み専用です。

構文

[*formname.*]**MaxButton**

対象

フォーム

解説

MaxButton プロパティは、デザイン時にフォーム デザイナで True (表示する) または False (表示しない) に設定できます。実行時には変更できません。デフォルトは True です。

最大化ボタンを押すと、フォーム ウィンドウは拡大されて画面全体に表示されます。フォームが最大化されると、最大化ボタンは自動的に復元ボタンに変わります。フォームが復元されると、復元ボタンは自動的に最大化ボタンに戻ります。

実行時にフォームを最大化または復元すると、Resize および Paint イベントが発生します。

MaxButton プロパティが True (表示する) に設定されていても、フォームの BorderStyle プロパティが 0 (なし) に設定されていると、最大化ボタンは表示されません。

WindowState プロパティは、ウィンドウの表示状態を決めます。実行時にコード中からウィンドウを最大化したいときは、WindowState プロパティを 2 (最大化されている) に設定してください。

参照

MinButton プロパティ

MID$ 関数, MID$ ステートメント

機能

MID$** 関数は、文字列の中から、指定したバイト数分の文字列を返す文字列処理関数です。**MID$ ステートメントは、文字列変数の一部を別の文字列に置き換える文字列処理ステートメントです。

構文

**MID$**(*string-expression$*, *start%* [,*length%*])

**MID$**(*stringvariable$*, *start%* [,*length%*]) = *string-expression$*

string-expression$	元となる文字列。
start%	文字列を取り出す開始位置を先頭からのバイト数で示す、1～32,767 の整数。
length%	部分文字列のバイト数を示す、1～32,767 の整数。省略すると、*start%* が示す位置以降の、元となる文字列の最後までと見なされます。
stringvariable$	新しく置き換える文字列。文字列変数，文字列定数，文字列式のいずれでもかまいません。

解説

MID$** 関数，**MID$ ステートメントともに、引数はバイト数で指定することに注意してください。元の文字列が 2 バイト文字を含む場合、指定によっては文字の途中からが処理の対象になることがあります。文字単位で指定するには、**KMID$** 関数およびステートメントを使ってください。

引数 *start%* および *length%* は、1～32,767 の値を持つ数式でなければなりません。

MID$ 関数で、*start%* で指定した位置が、*string-expression$* で指定して文字列の最後より後ろを示す場合は、null 文字列を返します。また、*start%* で指定した位置以降の文字列の長さが、*length%* で指定した長さより短ければ、以降の文字列全体を返します。

string-expression$ のバイト数を調べるには **LEN** 関数を使います。

MID$ ステートメントで、*start%* が示す位置以降の長さが *stirng-expression$* の長さより短ければ、余りの部分は置き換えられません。*length%* の指定にかかわらず、置き換え後に *stringvariable$* の長さが長くなることはありません。

length% を省略すると、*start%* が示す位置以降のすべての部分と見なされます。

Min プロパティ

Max プロパティを参照してください。

MinButton プロパティ

機能　フォームの右上隅に最小化ボタンを表示するかどうかを決めます。実行時には読み込み専用です。

構文　[*formname.*]**MinButton**

対象　フォーム

解説　MinButton プロパティは、デザイン時にフォーム デザイナで True (表示する) または False (表示しない) に設定できます。実行時には変更できません。デフォルトは True です。

最小化ボタンを押すと、フォーム ウィンドウは縮小されてアイコンになります。

実行時にフォームを最小化すると、Resize イベントが発生します。

MinButton プロパティが True (表示する) に設定されていても、フォームの BorderStyle プロパティが 0 (なし) に設定されていると、最小化ボタンは表示されません。

WindowState プロパティは、ウィンドウの表示状態を決めます。実行時にコード中からウィンドウを最小化したいときは、WindowState プロパティを 1 (最小化されている) に設定してください。

参照　MaxButton プロパティ

MINUTE 関数

機能　日付と時刻のシリアル値を評価して、0～59 までの範囲で、何分かを返します。

構文　**MINUTE**(*serial*)

serial　日付と時刻を表すシリアル値 (-53,688～65,380)。

MKDIR ステートメント

CHDIR ステートメントを参照してください。

MK*x*$ 系関数 (MKC, MKI, MKL, MKS, MKD)

機能

FIELD および **PUT** ステートメントと組み合わせて、数値をランダムアクセス ファイルに書き込むために使います。整数 (**MKI**)，長整数 (**MKL**)，単精度浮動小数点数 (**MKS**)，倍精度浮動小数点数 (**MKD**)，通貨型数 (**MKC**) を、数値内部表現の文字列にそれぞれ変換します。

構文

MKI$(*numeric-expression%*)

MKL$(*numeric-expression%*)

MKS$(*numeric-expression%*)

MKD$(*numeric-expression%*)

MKC$(*numeric-expression%*)

numeric-expression%　任意の数式。

解説

これらの関数は、以前のバージョンと互換性を保つために残されています。ユーザー定義型変数を使うと、ランダムアクセス ファイルへの入出力が、より効率的にできます。

参照

CV*x* 系関数，**FIELD** ステートメント，**MK***x***MBF$** 系関数，**PUT** ステートメント (ファイル I/O)

MK*x*MBF$ 系関数 (MKSMBF$, MKDMBF$)

CV*x***MBF** 系関数を参照してください。

MONTH 関数

機能

日付と時刻のシリアル値を評価して、1～12 までの範囲で、何月かを返します。

構文

MONTH(*serial*)

serial　日付と時刻を表すシリアル値 (-53,688～65,380)。

MouseDown, MouseUp イベント

機能

ユーザーが右または左マウス ボタンを押したとき (MouseDown)、または離したとき (MouseUp) に、そのときフォーカスを持っているオブジェクトに対して発生します。

構文

SUB Form_{MouseDown | MouseUp} (*Button* AS INTEGER, *Shift* AS INTEGER, *X* AS SINGLE, *Y* AS SINGLE)

SUB *ctlname*_{MouseDown | MouseUp} ([*Index* AS INTEGER,] *Button* AS INTEGER, *Shift* AS INTEGER, *X* AS SINGLE, *Y* AS SINGLE)

Index	コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。
Button	左右どちらのボタンが操作されたかを示す値が渡されます。
Shift	イベント発生時の [SHIFT], [CTRL], [メニュー] の各キーの状態を示す値が渡されます。
X, Y	そのときのマウス ポインタの、発生したイベントを認識したコントロールに対する相対位置が渡されます。

対象

ディレクトリ リスト ボックス，ファイル リスト ボックス，フォーム，ラベル，リスト ボックス，ピクチャー ボックス

解説

Click イベントや DblClick イベントと異なり、MouseDown および MouseUp イベントでは、マウスの左右のボタンを区別できます。

Button 引数には、左右どちらのボタンが操作されたかを示す値が渡されます。1 は左ボタン、2 は右ボタンを表します。

Shift 引数に指定した変数には、イベント発生時の [SHIFT], [CTRL], [メニュー] の各キーの状態を示す値が格納されます。[SHIFT] キーを 1，[CTRL] キーを 2，[メニュー] キーを 4 として、イベント発生時に押されていたキーの値を合計した値が渡されます。例えば、値が 7 であれば 3 つのキーすべてがいっしょに押されていたことを示し、値が 3 なら [SHIFT] キーと [CTRL] キーがいっしょに押されていたことを示します。

Shift 引数の値と MouseMove イベントを使うと、マウスとキーボードを組み合わせた操作に対応するコードを記述できます。

Click および DblClick イベントと MouseDown, MouseUp イベントの関係において、次の点に注意してください。

- マウス ポインタが、あるオブジェクトの上にあるときにマウス ボタンが押されると、そのオブジェクトが、MouseDown イベントを認識します。つまり MouseDown イベントが発生したときは、マウス ポインタは必ずそのオブジェクトが配置されている領域内にあります。それ以降、MouseUp イベントが発生するまで、すべてのマウスに関連するイベントは、そのオブジェクトによって認識されます。これは、マウスに関連するイベントの発生によって得られたマウス ポインタの X と Y 座標が、必ずしもそれを認識したオブジェクトが配置されている領域内にあるとは限らないことを意味します。ただし、X と Y 座標の値は、そのオブジェクトの Top および Left プロパティに対する相対値であることには変わりありません。
- マウス ボタンが連続して押された場合、最初の MouseDown イベントを認識したオブジェクトが、すべてのボタンが離されるまで、すべてのイベントを認識します。
- MouseDown イベントと MouseUp イベントの発生回数は必ずしも一致しません。マウス ボタンが短い間隔で続けて押された場合、最後に MouseUp イベントだけが発生します。

マウス ポインタがすでにそのオブジェクトの領域を離れている場合、*X* および *Y* 引数で、現在の位置を調べてください。

マウスの移動をイベントとして利用するには、MouseMove イベントを使います。MouseMove イベントの *Button* 引数の意味は、MouseDown および MouseUp イベントと異なることに注意してください。MouseDown や MouseUp イベントでは、*Button* は 1 (左) または 2 (右) のどちらかですが、MouseMove イベントでは、両方のボタンが押されたままマウスが移動すれば、*Button* に渡される値は 3 になります。

参照

Click イベント，DblClick イベント，MouseMove イベント，ScaleHeight プロパティ，ScaleWidth プロパティ

MouseMove イベント

機能

ユーザーがマウスを動かしたときに発生します。

構文

SUB Form_MouseMove (*Button* AS INTEGER, *Shift* AS INTEGER, *X* AS SINGLE, *Y* AS SINGLE)

SUB *ctlname*_MouseMove ([*Index* AS INTEGER,] *Button* AS INTEGER, *Shift* AS INTEGER, *X* AS SINGLE, *Y* AS SINGLE)

Index	コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。
Button	左右どちらのボタンが操作されたかを示す値が渡されます。
Shift	イベント発生時の [SHIFT], [CTRL], [メニュー] の各キーの状態を示す値が渡されます。
X, Y	そのときのマウス ポインタの、発生したイベントを認識したコントロールに対する相対位置が渡されます。

対象　ファイル リスト ボックス，フォーム，ラベル，リスト ボックス，ピクチャー ボックス，ディレクトリ リスト ボックス

解説　MouseMove イベントは、マウス ポインタがオブジェクト上を移動するときに連続的に発生します。

MouseMove イベントを認識するオブジェクトは、ボタンが押されたまま移動しているかどうか、つまりドラッグされているかどうかによって異なります。

ボタンが押されていない場合、あるオブジェクトが配置されている領域の上でマウス ポインタが移動すると、そのオブジェクトに対して MouseMove イベントが発生します。フォームに対する MouseMove イベントは、コントロールが配置されていない領域でマウスが動かされた場合に発生します。

ボタンが押されながら移動している場合、つまりすでに MouseDown イベントが発生している場合は、MouseDown イベントが発生したオブジェクトに対して、MouseMove イベントも発生します。これは、すでにマウス ポインタがそのオブジェクトの領域を離れていても同じです。何もコントロールが配置されていない領域でマウス ボタンが押され、移動してコントロールの上に来た場合は、フォームに対して MouseDown イベントが発生していると解釈され、そのコントロールは MouseMove イベントを認識しません。

マウス ポインタがすでにそのオブジェクトの領域を離れている場合、*X* および *Y* 引数で、現在の位置を調べてください。

参照　Click イベント，DblClick イベント，ScaleHeight プロパティ，ScaleWidth プロパティ

MousePointer プロパティ

機能　実行時に、フォームやコントロールの上にマウスがあるとき表示されるマウス ポインタの種類を決めます。

構文　{[*formname*.][*ctlname*.]|**SCREEN**.}**MousePointer**[= *char%*]

char%　マウス ポインタの形状を示す、0～12 の整数。デフォルトは 0 です。

対象　チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，**SCREEN** オブジェクト，テキスト ボックス，垂直スクロール バー

解説　MousePointer プロパティはマウス ポインタの形を制御します。マウス ポインタがフォーム上の各種のコントロールの上を通るとき、マウスの機能が変化していることをユーザーに知らせるために、このプロパティを使います。

有効な設定値は次のとおりです。

設定値	形状
0	矢印 (デフォルト)
1	小型矢印
2	クロス
3	I ビーム
4	二重四角
5	サイズ (四方矢印)
6	斜め矢印 (右上から左下)
7	上下サイズ (上下矢印)
8	斜め矢印 (左上から右下)
9	左右サイズ (左右矢印)
10	上矢印
11	砂時計
12	禁止マーク

SCREEN オブジェクトの MousePointer プロパティを設定すると、その設定がすべてのフォームおよびコントロールに対する設定よりも優先されます。**SCREEN** オブジェクトの MousePointer プロパティを 0 に設定すると、マウス ポインタの形は通過中のフォームやコントロールに対する設定に従います。

コントロールの MousePointer プロパティの値が 0 (デフォルト) のとき、形状はそのコントロールのコンテナ (フォーム，フレームまたはピクチャー ボックス) と同じになります。

参照 MouseMove イベント

MouseUp イベント

MouseDown イベントを参照してください。

MOVE メソッド

機能

フォームやコントロールの移動およびサイズ変更を行います。

構文

[{[*formname*.]*ctlname*.|*formname*.}]**MOVE** *left%*[,*top%*[,*width%*[,*height%*]]]

left%	左端の水平座標。
top%	上端の垂直座標。
width%	オブジェクトの幅。
height%	オブジェクトの高さ。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

MOVE メソッドで省略できない引数は *left%* だけですが、他の引数を指定するためには、指定しようとしている引数よりも構文中で前に位置する引数をすべて指定しなければなりません。例えば、*width%* を指定するためには *left%* と *top%* も指定する必要があります。残りの指定されなかった引数の値は変更されません。

フォームまたはコントロールが移動すると、Left, Top, Width, Height の各プロパティは **MOVE** メソッドの引数で指定された新しい値に自動的に設定されます。ただし、コンボ ボックス (Style = 0 または 2) およびドライブ リスト ボックスの Height の値は変更できません。これらのコントロールでは、Height プロパティは読み込み専用です。

座標はキャラクタ単位で表されます。画面 (**SCREEN** オブジェクト) 上でのフォームの移動やフォーム上でのコントロールの移動は、常に画面やコンテナの左上角を原点 (0, 0) とする相対値で示されます。通常、“コンテナ” はフォームですが、フレームやピクチャー ボックスもコンテナにすることができます。

非表示になっているフォームやコントロールも移動できます。例えば、そのとき表示されているピクチャー ボックスに、非表示になっている複数のコントロールが置かれている場合、ピクチャー ボックスの移動には長い時間がかかることがあります。このような場合、次のようなテクニックを使うと効果的です。

- コンテナであるピクチャー ボックスの Visible プロパティを False (0) に設定する。
- ピクチャー ボックスを移動する。
- Visible プロパティを True (-1) に戻す。

MSGBOX ステートメント, MSGBOX 関数

機能

MSGBOX ステートメントは、あらかじめ用意されている、コマンド ボタンを含むダイアログ ボックスを表示します。**MSGBOX** 関数は、同様にダイアログ ボックスを表示し、押されたキーを示す値を返します。グラフィックス モードでは実行できません。

構文

MSGBOX ステートメント

MSGBOX *msg$*[, *type%*[, *title$*]]

MSGBOX 関数

MSGBOX(*msg$*[, *type%*[, *title$*]])

msg$　ダイアログ ボックスにメッセージとして表示するテキストを示す文字列式。

type%　表示するボタンの個数と種類、および最初にフォーカスを持っているコマンド ボタンを示す値の合計。省略すると <了解> ボタンだけが表示されます。

title$　ダイアログ ボックスのタイトル バーに表示するタイトルを示す文字列式。省略するとタイトルは表示されません。

解説

次のようなコマンド ボタンの組み合わせが用意されています。また、ダイアログ ボックスが表示されたとき、どのコマンド ボタンがフォーカスを持っているかを指定できます。*type%* 引数には、この 2 つの値の合計を指定します。

値	意味
0	<了解> (デフォルト)
1	<了解>, <取消>
2	<A/中止>, <R/再試行>, <I/無視>
3	<Y/はい>, <N/いいえ>, <取消>
4	<Y/はい>, <N/いいえ>
5	<R/再試行>, <取消>
0	1 番目のボタンがフォーカスを持つ
256	2 番目のボタンがフォーカスを持つ
512	3 番目のボタンがフォーカスを持つ

いずれかのコマンド ボタンがクリックされるか、[ENTER] キーまたは [ESC] キー が押されると、ダイアログ ボックスはクローズします。また、コントロール メニューの [クローズ] コマンドでもクローズできます。

MSGBOX 関数では、押されたコマンド ボタンを示す次のような値が返されます。

値	押されたボタン
1	<了解>
2	<取消>
3	<A/中止>
4	<R/再試行>
5	<I/無視>
6	<Y/はい>
7	<N/いいえ>

[ESC] キーが押されると、<取消> ボタンが押されたと見なされ、2 が返されます。

MSGBOX ステートメントは値を返さないため、*type%* 引数に 0 を指定するか省略して、<了解> ボタンだけを持つダイアログ ボックスを表示するときにだけ使うようにしてください。逆に、<了解> ボタンだけを持つダイアログ ボックスを表示するときは、どのボタンが押されたかを調べる必要がないため、**MSGBOX** 関数を使うことは意味がありません。

ダイアログ ボックスは、モーダルなダイアログ ボックスとして表示されます。このダイアログ ボックスが表示されている間は、他のフォームに対するイベントは発生しません。

このダイアログ ボックスは移動できますが、サイズは変更できません。

1 行に表示可能なメッセージは最大 76 バイトです。*msg$* 引数に 76 バイトを超える長さの文字列を指定すると、スペースまたは 2 バイト文字の前後で自動的に改行されます。76 バイト以下の長さで意図的に改行したいときは、次のように、*msg$* 引数に指定する文字列の途中に改行コード (CHR$(13) と CHR$(10)) を挿入してください。複数行に渡る場合のメッセージの最大長は 1024 バイトです。メッセージがこれより長いとき、超える部分は切り捨てられます。

```
msg$ = "この行が、メッセージの 1 行目です。"
msg$ = msg$ + CHR$(13) + CHR$(10) + "これが 2 行目です。"
x% = MSGBOX (msg$)          ' MSGBOX 関数の場合
```

タイトル バーに表示可能なタイトルは最大 74 バイトです。タイトルがこれより長いとき、超える部分は切り捨てられます。

ダイアログの幅は、次の 3 つの幅のうちで最も大きなものを基準に決まります。

- メッセージの幅 (最大 76 バイト分)
- タイトルの幅 (最大 74 バイト分)
- ボタンを表示するのに必要な幅

参照　**INPUTBOX$** 関数

MultiLine プロパティ

機能

テキスト ボックスに複数行のテキストを表示できるかどうかを決めます。実行時には読み込み専用です。

構文

[*formname.*]*ctlname*.**MultiLine**

ctlname	テキスト ボックスの名前。

対象

テキスト ボックス

解説

MultiLine プロパティは、デザイン時にフォーム デザイナで True (-1) または False (0) に設定できます。実行時には変更できません。デフォルトは False (0) です。

MultiLine プロパティが True (-1) に設定されているテキスト ボックスでは、テキスト ボックスの幅を超えるテキストを入力または設定したとき、自動的にワード ラップされます。False (0) に設定されている場合、横スクロールして表示されます。[ENTER] キーを入力しても無視されます。

ScrollBars プロパティを使って、テキスト ボックスにスクロール バーを加えることもできます。

デフォルト ボタン (フォーカスがなくても [ENTER] キーが押されたときに Click イベントが発生するコマンド ボタン) を持たないフォーム上では、MultiLine プロパティが True (-1) のテキスト ボックス内で [ENTER] キーを押すと、テキストは改行されてカーソルは次の行に移動します。デフォルト ボタンがあるときは、[ENTER] はデフォルト ボタンのクリックの意味になります。次の行に移動するには [CTRL]+[ENTER] キーを押す必要があります。MultiLine プロパティが False (0) のテキスト ボックス内では、デフォルト ボタンのあるなしにかかわらず、[ENTER] キーは常に無視されます。

参照

ScrollBars プロパティ

NAME ステートメント

機能

ファイル名を変更します。また、同じドライブのディレクトリ間でのファイルの移動もできます。

構文

NAME *oldfilespec$* **AS** *newfilespec$*

oldfilespec$	元のファイル名を表す文字列式。
ewfilespec$	新しいファイル名を表す文字列式。

解説

NAME ステートメントは、MS-DOS の RENAME コマンドと同様の動作をします。さらに、指定したファイルを同じドライブの別のディレクトリに移動させることもできます。

oldfilespec$* と *newfilespec$ には、元のファイルと新しいファイルをそれぞれ指定します。*oldfilespec$* で指定するファイルは既存のファイルでなければならず、逆に *newfilespec$* にすでに存在するファイルを指定することはできません。*newfilespec$* に *oldfilespec$* と異なるディレクトリを指定すると、そのファイルを (*newfilespec$* で指定した) 別のディレクトリに移動することができます。ただしこの場合でも、別のドライブに対して移動することはできません。*newfilespec$* と *oldfilespec$* に別のドライブを指定すると、"ディスク間のリネームはできません" エラーになります。また、ディレクトリを移動することはできません。

そのときオープンされているファイルに対して **NAME** ステートメントを実行すると、"すでにオープンされています" エラーになります。ファイル名を変える前に、オープンされているファイルをクローズしてください。

NEWPAGE メソッド

機能

PRINTER オブジェクトに現在出力中のページの終了を知らせ、次のページに進みます。

構文

PRINTER.NEWPAGE

対象

PRINTER オブジェクト

解説

NEWPAGE メソッドは、**PRINTER** オブジェクトに CTRL+L (16 進数の 0C, フォームフィード) を送ります。プリンタは、これを受けて改ページを行います。

出力は、**PRINTER** オブジェクトの PrintTarget プロパティに設定されているプリンタまたはファイルに送られます。

参照

ENDDOC メソッド, **PRINT** メソッド

Normal プロパティ

Archive プロパティを参照してください。

NOW 関数

機能 現在のシステムの日付と時刻を表すシリアル値を返します。

構文 **NOW**

OCT$ 関数

HEX$ 関数を参照してください。

ON ERROR ステートメント

機能 エラー処理ルーチンを設定してエラー トラップを有効にする、エラー処理ステートメントです。

構文 **ON [LOCAL]ERROR {GOTO** *line***|RESUME NEXT|GOTO 0}**

LOCAL	エラー処理ルーチンが、それの定義されているプロシージャに対して "ローカル" であることを示します。省略すると、モジュール レベルのエラー処理ルーチンであると解釈されます。
GOTO *line*	*line* で指定した行から始まる、エラー処理ルーチンを有効にします。*line* には、先頭行を示す行ラベルまたは行番号を指定します。プログラム実行中にエラーが発生すると、ここで設定したエラー処理ルーチンに制御が移ります。 *line* で指定する行は、モジュールレベル コードか、(**LOCAL** キーワードが指定されていれば) 同じプロシージャ内になければなりません。このどちらにも *line* が見つからないときは、"ラベルが未定義です " エラーになります。
RESUME NEXT	エラーが発生してもプログラムを中断せず、エラーが発生したステートメントの次のステートメントから実行を続けることを示します。エラーに対するエラー コードを得るには、**ERR** 関数を使ってください。
GOTO 0	エラーが発生しても、トラップされないようにします。そのモジュールのモジュールレベル コードに含まれるすべてのエラー処理ルーチン、または (**LOCAL** キーワードを指定しているときは) そのプロシージャに含まれるすべてのエラー処理ルーチンを無効にします。

解説 **ON ERROR** ステートメントによってエラー処理の方法が設定されていなければ、プログラム実行中にエラーが発生すると、エラー メッセージが表示されてプログラムの実行は停止します。エラー処理ルーチンが設定され、エラー トラップが有効になっていれば、実行時エラーが発生しても、プログラムの制御はエラー処理ルーチンに移り、

そのエラー処理ルーチンは "アクティブ" になります。実行時エラーがトラップされたときから、**RESUME** ステートメントが実行されるまで、エラー処理ルーチンはアクティブです。

LOCAL キーワードを指定すると、エラー処理ルーチンが、それを含むプロシージャに対してローカルであることを示します。ローカルなエラー処理ルーチンには、次のような特長があります。

- 有効になっているモジュール レベルのエラー処理ルーチンより優先します。
- そのエラー処理ルーチンを含むプロシージャが実行されている間だけ、有効となります。

ローカルなエラー処理ルーチンは、それを含むプロシージャから直接または間接的に呼ばれているプロシージャを実行している間も、有効のままです。これらの (呼び出された) プロシージャのどれかが、別のエラー処理ルーチンを設定することもあります。この場合、そちらのエラー処理ルーチンが優先されます。

有効なエラー処理ルーチンがないプロシージャやモジュールでエラーが発生すると、Basic はエラーをトラップするために、有効でありアクティブでないエラー処理ルーチンを探します。エラーの発生したプロシージャを呼び出した、すべてのプロシージャとモジュールが順に検索されます。複数のモジュールで構成されているプログラムでは、すべてのモジュールに対して、対応するモジュール レベルのエラー処理ルーチンが検索されます。

引数 *line* は、そのモジュールレベル コードに含まれる行を示していなければなりません。**LOCAL** キーワードを指定したときは、さらに同じプロシージャに含まれる行でなければなりません。条件が満たされないと、コンパイル時に "ラベルが未定義です" エラーが発生します。

RESUME NEXT を指定すると、エラーが発生しても実行を中断せず、エラーが発生したステートメントの次のステートメントから実行を続けることができます。次のいずれかの場合に **RESUME NEXT** を使ってください。

- 実行時にエラーが発生したブロックの残りの部分を実行し、その後でエラーの原因をチェックしたいとき。
- エラー処理ルーチンをプログラムの独立した位置に記述せずに、メイン プログラムやプロシージャ中に作成する場合。

GOTO 0 を指定すると、エラーが発生してもトラップされないようになります。この指定は行番号 0 の位置にエラー処理ルーチンがあることを示しているわけではありません。プログラムやプロシージャに行番号 0 が含まれていても、その位置へ制御が移ることはありません。

エラー処理ルーチンは、**SUB** および **FUNCTION** プロシージャや **DEF FN** 関数のように明確な構造を持つわけではなく、行ラベルや行番号で始まり **RESUME** ステートメントで終わる、単なるコード ブロックであることに注意してください。

エラー処理ルーチンは次の条件がなければ実行されません。

- **ON ERROR GOTO** *line* ステートメントが実行されて、それが有効になっている。
- プログラムの構造上の問題によって、誤って制御がエラー処理ルーチンに移るか、実行がエラー処理ルーチンに達してしまった。
- **ERROR** ステートメントによって意図的にエラーを発生させた。

エラー処理ルーチンの中でエラーの原因を調べるためには、**ERR** 関数を使います。**ERR** 関数は、最後に発生したエラーのエラーコードを返します。したがってエラー処理ルーチンでは、他のエラーが発生する可能性のある処理やルーチンを呼び出す前に、この値をテストするか別に保存するようにしてください。

エラー処理ルーチンを含むプログラムを BC コマンドラインからコンパイルするときは、必ず /E(On Error) または /X(Resume Next) オプションを指定してください。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

参照　**ERL** 関数，**ERR** 関数，**ERROR$** 関数

ON *event* 系ステートメント (ON {COM | KEY | PLAY | TIMER | UEVENT})

機能　イベントがトラップされたときに分岐する処理ルーチンを設定します。フォームやコントロールに関連したイベントには影響を与えません。

構文

ON COM(*port%*) **GOSUB** {*linenumber*|*linelabel*}

ON KEY(*keyno%*) **GOSUB** {*linenumber*|*linelabel*}

ON PLAY(*queuemin%*) **GOSUB** {*linenumber*|*linelabel*}

ON TIMER(*sec&*) **GOSUB** {*linenumber*|*linelabel*}

ON UEVENT GOSUB {*linenumber*|*linelabel*}

port%	1 または 2 のシリアル ポート番号。
keyno%	ファンクション キー，方向キー，ユーザー定義キーのキー番号を示す、次のような整数。
queuemin%	BGM キュー内の最小残り音符数を示す、 1～32 の整数。
sec&	イベントが発生するまでの秒数を示す、 1～86,400 の整数。
linenumber	イベントが起こったときに実行される、イベント処理ルーチンの最初の行を示す行番号。
linelabel	イベントが起こったときに実行される、イベント処理ルーチンの最初の行を示す行ラベル。

解説

ON *event* 系のステートメントは、イベントが発生したときに分岐する、イベント処理ルーチンの開始行を指定します。その行は、モジュールレベルコード中になければなりません。0 を指定すると、イベントがトラップされなくなります。0 を、ルーチンの開始行として指定することはできません。

イベントトラップの有効/無効は、次のようなステートメントで設定します。

ステートメント	説明
event **ON**	イベントトラップを有効にします。 *event* **ON** ステートメントを実行したあとでなければ、イベントトラップは行われません。
event **OFF**	イベントトラップを無効にします。次に *event* **ON** ステートメントを実行するまで、イベントトラップは行われません。無効のときに発生したイベントは無視されます。
event **STOP**	イベントトラップを中断します。次に *event* **ON** ステートメントを実行するまでイベントはトラップされませんが、中断状態のときに発生したイベントは記憶されています。 *event* **ON** ステートメントによってトラップが有効になると、すぐにトラップされます。

イベント トラップが有効であるとき、そのイベントが発生していないかどうかが自動的に監視されています。このチェックは、プログラミング環境で実行されている場合と、/V オプション付きでコンパイルされた実行可能ファイルでは、ステートメントごとに行われます。/W オプション付きでコンパイルされた実行可能ファイルでは、行番号または行ラベルごとにチェックされます。イベントが発生していると、**ON** *event* 系ステートメントで設定されているイベント処理ルーチンに制御が移ります。

イベント トラップを行うステートメントを含むプログラムを BC コマンドラインからコンパイルするとき、/W または /V オプションを付けずにコンパイルすると、"ON event ステートメントを含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

イベントがトラップされ、対応する処理ルーチンに制御が移ると、再帰的なトラップが起こらないように自動的に *event* **STOP** が実行されます。処理ルーチンが終了すると、そのルーチンの中で明確に *event* **OFF** が実行されていない限り、自動的に *event* **ON** ステートメントが実行されます。

イベント処理ルーチン終了時に指定した行に制御が戻るように、**RETURN** *linenumber* または **RETURN** *linelabel* の形式を使うことができます。しかし、この形式の **RETURN** ステートメントを使うときには、次の点に注意してください。

ループ、サブルーチンなどの実行中にイベントがトラップされた場合、適切な行に制御を戻さなければ、"FOR がありません" などのエラーになります。

プロシージャの実行中にイベントがトラップされた場合、**RETURN** *linenumber*, **RETURN** *linelabel* ステートメントは、元のプロシージャに制御を戻すことができません。これは、指定する行番号や行ラベルは、モジュールレベル コード内になければならないためです。

ON COM(*port%*) の使い方

通信ポートが文字を受け取ったときに分岐する行を設定します。*port%* 引数には 1 または 2 のシリアルポート番号を指定します。

ON KEY(*keyno%*) の使い方

指定したキーが押されたときに分岐する行を設定します。*keyno%* 引数には、ファンクション キー，方向キー，ユーザー定義キーのキー番号を示す、次のような整数を指定します。

値	キー
1～10	ファンクション キー [F1] ～ [F10]
11	[↑] キー
12	[←] キー
13	[→] キー
14	[↓] キー
15～25	ユーザー定義キー
26～31	ファンクション キー [F11] ～ [F16] (機種によっては無効です)

26～31 に割り当てられているファンクション キーは、ハードウェアによっては無効です。30, 31 は、QuickBasic や Microsoft Basic と意味が異なりますので注意してください。

ON KEY ステートメントは、フォームが表示されている状態では利用できません。

ON PLAY(*queuemin%*) の使い方

BGM キューの中にある、バックグラウンドで演奏される音符の残りが *queuemin%* 引数で指定した数より少なくなったときに分岐する行を設定します。これによって、バックグラウンドで音楽を演奏しながらプログラムを実行できます。

queuemin% には、BGM キュー内の最小残り音符数を示す、1～32 の整数を指定します。残りの音符の数が、*queuemin%* から *queuemin%*-1 になったときにイベントが発生します。

ON PLAY ステートメントを使うときは次の点に注意してください。

- トラップは、バックグラウンドで演奏されているときにだけ起こり、フォアグラウンドで演奏されているときには起こりません。
- **PLAY ON** ステートメントが実行されたとき、BGM キューの音符数がすでに *queuemin%*-1 以下であると、トラップされません。
- *queuemin%* の値が大きいと、頻繁にトラップが起こり、プログラムの実行速度が低下します。

ON TIMER(*sec&*) の使い方

sec& 引数で指定した秒数が経過したときに分岐する行を設定します。*sec&* には 1～86,400 (1 秒～24 時間) が指定できます。

ON UEVENT の使い方

ユーザー定義のイベントが発生したときに分岐する行を設定します。通常はハードウェア割り込みで使います (ソフトウェア割り込みでも使えます)。

次に示すコードが必要です。

- 割り込みサービス ルーチン
- 割り込みサービス ルーチンのアドレスを割り込みベクタ テーブルに登録する、初期化ルーチン
- 割り込みサービス ルーチンが集めたデータを処理するために Basic のプログラムが呼び出すルーチン

初期化ルーチンが、別の割り込み処理ルーチンが使うアドレスを利用するときは、プログラム終了時にアドレスを元に戻してください。

これらのルーチンは通常、アセンブリ言語で書きます。しかし、コンパイラが割り込みサービス ルーチンを生成でき、オブジェクト コードを Basic プログラムとリンクできれば、どんな言語を使ってもかまいません。

次の手順で、ユーザー定義イベントを処理してください。

- イベント処理ルーチンを書き Basic のプログラムに追加します。
- **ON UEVENT** ステートメントで、ユーザー定義イベントのイベント処理ルーチンを設定します。
- **UEVENT ON** ステートメントで、ユーザー定義イベントのトラップを有効にします。
- 割り込み初期化ルーチンを呼び出し、割り込みサービス ルーチンのアドレスを割り込みベクタ テーブルに登録します。

割り込みが起こると、制御は割り込みサービス ルーチンに移ります。割り込みサービス ルーチンは、必要なデータを集めて保存してから、SetUEvent ルーチン (プロシージャ) を呼び出します。SetUEvent ルーチンは Basic の次のステートメントが実行される前に、Basic がチェックしたフラグをセットします (/V オプションの代わりに /W オプションを使ってコンパイルされたコードを実行するときはラベル)。フラグがセットされると、制御は **ON UEVENT GOSUB** で設定したイベント処理ルーチンに移ります。

SetUEvent ルーチンは Basic の一部で、プログラムをコンパイルするか、/L オプションを指定して VBDOS.EXE を起動すると、自動的に Basic の中に組み込まれます。割り込みサービス ルーチンは、SetUEvent ルーチンを呼び出さなければなりません。これは、イベントが起こったことをプログラムに知らせる唯一の方法です。SetUEvent ルーチンは、アセンブリ言語だけではなく、どんな言語プログラムからでも呼び出せます。

SetUEvent ルーチンは関数ではないので、Basic に値を返すことはありません。戻り値が必要なときは、関数を呼び出す Basic のプログラムを書いてください (通常、割り込みサービス ルーチンによって呼び出されるようにします)。この関数は、**DECLARE** ステートメントで宣言してください。

ON UEVENT GOSUB ステートメントは、イベント処理ルーチンとユーザー定義イベントを結び付けますが、トラップを有効にするわけではありません。ユーザー定義イベントのトラップを有効または無効にするには、**UEVENT** ステートメントを使ってください。

参照

EVENT ステートメント，**ON COM** ステートメント，**ON KEY** ステートメント，**ON PLAY** ステートメント，**ON TIMER** ステートメント，**ON UEVENT** ステートメント

ON...GOSUB, ON...GOTO ステートメント

機能

式の値に応じてプログラムの制御を分岐させる、フロー制御ステートメントです。

構文

ON *expression%* **GOSUB** *line-label-list*

ON *expression%* **GOTO** *line-label-list*

expression%　分岐先を決めるための整数。整数以外を指定すると、整数に丸められます。

line-label-list　カンマ (,) で区切った、分岐先の行を示す行ラベルまたは行番号。行ラベルと行番号のどちらを使ってもかまいません。また混在させることもできます。指定できる行ラベルや行番号の最大数は 60 です。

解説

複数分岐を行うには、**SELECT CASE** ステートメントを使う方がより強力かつ柔軟な方法です。**ON...GOSUB** や **ON...GOTO** ステートメントの代わりに、**SELECT CASE** ステートメントを使うことをおすすめします。

expression の値は、1 以上で、*line-label-list* に指定された行番号や行ラベルの数以下でなければなりません。これ以外の値であると、次のように処理されます。

値	結果
0 または指定した行番号や行ラベルの数より大きな数	次のステートメントへ制御が移ります。
負の数または 255 より大きな数	"引数が有効範囲にありません " エラーが発生します。

参照

SELECT CASE ステートメント

OPEN ステートメント (ファイル I/O)

機能

ファイルまたはデバイスをオープンし、入出力を可能にします。

構文

OPEN *file$*[**FOR** *mode*][**ACCESS** *access*][*lock*] **AS** [#]*filenumber%*[**LEN=***reclen%*]

file$	オープンするファイル名。ファイル名の指定に、ドライブやディレクトリの指定も含めることができます。デバイスをオープンするときはデバイス名 (KYBD:, SCRN:, COM*n*:, LPT*n*:, CONS:) を指定します。
mode	ファイル モードを示すキーワード (**RANDOM, BINARY, INPUT, OUTPUT, APPEND**)。
access	オープンするファイルに対して行う処理を示すキーワード (**READ, WRITE, READ WRITE**)。
lock	ロックのタイプを示すキーワード (**SHARED, LOCK READ, LOCK WRITE, LOCK READ WRITE**)。ネットワーク環境だけで使います。
filenumber%	ファイル番号を示す 1 ～ 255 の整数。この番号はオープンしたファイルまたはデバイスに割り当てられ、以降これらを指定するためには、このファイル番号を使います。すでに使われている番号は指定できません。
reclen%	32,767 以下の整数。ランダムアクセス ファイルの場合はバイト単位のレコード長を示し、省略すると 128 と見なされます。シーケンシャル ファイルの場合はバイト単位のバッファの容量を示し、省略すると 512 と見なされます。バイナリファイルの場合は指定する必要はありません。

解説

ファイルやデバイスに対して入出力を行う前には、その対象を必ず **OPEN** ステートメントでオープンしなければなりません。**OPEN** ステートメントは、そのファイルまたはデバイス用の入出力バッファを割り当て、そのバッファに使用されるアクセス モードを決めます。

file$ 引数には、MS-DOS の名前付け規約に従ってファイル名を指定します。ドライブやディレクトリの指定を含めることもできます。また、Visual Basic がサポートしているデバイスの名前も記述できます。

FOR キーワードに続けて指定する *mode* 引数には、ファイルのモードを示す、次のいずれかのキーワードを指定します。省略すると **RANDOM** と見なされます。

キーワード	機能
RANDOM	ランダムアクセス ファイル モード (デフォルト)。**OPEN** ステートメントが実行されたとき、**ACCESS** キーワードが省略されている場合は、次の順にファイルをオープンしようとします。 1. 読み込み/書き込み (**READ WRITE**) 2. 書き込み専用 (**READ**) 3. 読み込み専用 (**WRITE**)
BINARY	バイナリ ファイル モード。バイナリ モードでは、**GET** および **PUT** ステートメントを使って、ファイル内の指定したバイト位置で情報を読み書きできます。**ACCESS** キーワードが省略された場合の扱いは、**RANDOM** と同じです。
INPUT	シーケンシャル入力モード。シーケンシャル ファイルから読み込みだけを行います。
OUTPUT	シーケンシャル出力モード。シーケンシャル ファイルに書き込みだけを行います。
APPEND	シーケンシャル出力モード。ファイル ポインタはファイルの最後に設定され、レコード番号はファイルの最後のレコードに設定されます。**PRINT #** または **WRITE #** ステートメントで追加書き込みを行います。

INPUT, RANDOM, BINARY の各モードでは、1 つのファイルを同時に複数のファイル番号でオープンできます。一方、**OUTPUT, APPEND** モードでは、ファイルをいったんクローズしなければ、別のファイル番号で再びオープンすることはできません。

ACCESS キーワードに続けて指定する *access* 引数には、オープンしたファイルに対して行う操作を示す、次のいずれかのキーワードを指定します。ファイルが別のプロセスですでにオープンされていて、指定したアクセスが実行できないとき、**OPEN** ステートメントはファイルをオープンできず、"書き込みできません" エラーになります。**ACCESS** キーワードは、ネットワークをサポートしている、MS-DOS バージョン 3.1 以上を使っているときだけ有効です。ネットワークでロック操作を行うには、あらかじめ SHARE.EXE を実行しておく必要があります。バージョン 3.1 より前の MS-DOS で **ACCESS** キーワードを指定すると、"この機能はサポートされていません" エラーになります。

キーワード	機能
READ	読み込み専用。
WRITE	書き込み専用。
READ WRITE	読み込み/書き込み。このモードは、ランダムアクセス ファイル (**RANDOM**)，バイナリ ファイル (**BINARY**)，追加 (**APPEND**) モードでオープンしたシーケンシャル ファイルにだけ指定できます。

lock 引数は、ネットワークのように複数のプロセスが同時に処理を行う環境だけで働き、オープンファイルへの他のプロセスのアクセスを制限します。次に示すキーワードのいずれかを指定します。

ロックの種類	機能
SHARED	どのマシン上のプロセスからでも、ファイルを読み書きすることができます。ロックの種類を指定する **SHARED** キーワードを、**SHARED** ステートメントや他のステートメントの中で使う SHARED と混同しないでください。
LOCK READ	ファイルに対して最初に読み込みアクセスを行ったプロセスだけが、データの読み込みを行えます。その他のプロセスは、このファイルからデータを読み込むことができません。
LOCK WRITE	ファイルに対して最初に書き込みアクセスを行ったプロセスだけが、データの書き込みを行えます。その他のプロセスは、このファイルにデータを書き込むことができません。
LOCK READ WRITE	ファイルに対して最初に読み込みアクセスか書き込みアクセスを行ったプロセスだけがファイルの読み書きを行えます。その他のプロセスは、このファイルを読み書きできません。 **LOCK READ** か **LOCK WRITE** がすでに指定されているときは、これらが優先されます。

ロックの種類を指定しないと、現在のプロセスは、読み書きを行うために何度でもファイルをオープンすることができますが、他のプロセスは、ファイルがオープンしている間、アクセスすることができません。

前のプロセスがロックしたファイルを、**OPEN** ステートメントでオープンしようとすると、"書き込みできません" エラーになります。

filenumber% 引数には、ファイル番号を示す 1～255 の整数を指定します。オープンしたファイルは、指定したファイル番号と結び付けられます。他のファイル I/O ステートメントは、そのファイルにアクセスするときにこのファイル番号を使います。

reclen% 引数には、ランダムアクセス ファイルのレコード長、またはシーケンシャル ファイルのバッファ サイズをバイト単位で示す、32,767 以下の整数を指定します。ランダムアクセス ファイルの場合のデフォルトは 128、シーケンシャル ファイルの場合のデフォルトは 512 です。シーケンシャル ファイルで、バッファのサイズを大きくすると、Basic が使うことのできるメモリは少なくなりますが、ファイル I/O の速度は速くなります。バッファのサイズを小さくすると Basic が使うことのできるメモリは多くなりますが、ファイル I/O の速度は遅くなります。

シーケンシャル ファイルの場合、ファイル中のレコードサイズは一定でないため、*reclen%* の値は個々のレコード サイズと一致している必要はありません。

LEN=*reclen%* は、ファイル モードがバイナリ (**BINARY**) のときは無視されます。

Visual Basic は次のデバイスをサポートしています。引数 *file* にデバイス名を指定すると、これらのデバイスをファイルとしてオープンできます。

KYBD: (キーボード), SCRN: (ディスプレイ), COM*n*: (シリアル ポート), LPT*n*: (プリンタ ポート), CONS: (コンソール)

Visual Basic のファイル入出力システムでは、ユーザーがインストールしたデバイスも利用できます (キャラクタ デバイスについては、MS-DOS のマニュアルを参照してください)。

キャラクタ デバイスは、ディスク ファイルと同じ方法でオープンして利用できます。ただしディスク ファイルと違い、キャラクタはバッファに読み込まれません。デバイス ファイルのレコード長は 1 に設定されています。

Visual Basic は、行末でキャリッジ リターンだけを送ります。デバイスがライン フィードを必要とするときは、デバイス ドライバがそれを提供する必要があります。デバイス ドライバを作成するとき、他の Visual Basic のユーザーもデバイス制御データを読み書きする可能性があることに注意してください。デバイス制御データを読み書きするには、**IOCTL** ステートメントと **IOCTL$** 関数を使います。

Visual Basic のデバイスは、どれもバイナリ モードを直接はサポートしていません。ただし、ライン プリンタ デバイスは、次のように BIN キーワードを指定するとバイナリモードでオープンできます。プリンタを BIN モードでオープンすると、行末でキャリッジ リターンを出力しなくなります。

```
OPEN "LPT1:BIN" FOR OUTPUT AS #1
```

参照　**OPEN COM** ステートメント，**CLOSE** ステートメント，**FREEFILE** 関数，**TYPE** ステートメント

OPEN COM ステートメント

機能　データの入出力を行うため、通信チャネルをオープンして初期化します。

構文　**OPEN "COM***n*: *optlist1 optlist2*"[**FOR** *mode*] **AS** [#]*filenum*%[**LEN** = *reclen*%]

n	オープンするシリアル ポート (1 または 2)。
optlist1	通信速度，パリティ，データ長，ストップ ビットを示す通信パラメータ。デフォルトは、300bps の 7 ビットデータの転送に設定されています (バイナリデータの転送ではありません)。
optlist2	10 個までの、追加の通信パラメータ。
mode	入出力のモードを示す、**OUTPUT, INPUT, RANDOM** のいずれかのキーワード。デフォルトは **RANDOM** です。

filenum%	ファイル番号を示す 1～255 の整数。この番号は、オープンしたシリアル ポートに割り当てられます。以降これらを指定するためには、このファイル番号を使います。すでに使われている番号は指定できません。
reclen%	ランダムアクセス モードのバッファの大きさ (バイト単位)。デフォルトは 128 です。

解説

RS-232C インターフェイスを使って通信を行うには、通信に使うデバイス (シリアル ポート) を **OPEN COM** ステートメントでオープンしてください。

optlist1 引数には、次の構文に従って通信ラインのパラメータを指定します。

[*speed*][, [*parity*][, [*data*][, [*stop*]]]]

オプション	機能
speed	ボーレート (bps) を指定します。300, 1200, 2400, 4800, 9600 のいずれかを指定できます。デフォルトは 300 です。
parity	パリティチェックの方法を指定します。N(なし), E(偶数), O(奇数) のいずれかを指定できます。
data	データ長を指定します。7 または 8 が指定できます。デフォルトは 7 です。
stop	ストップ ビットを指定します。1 または 2 が指定できます。

各オプションは、この順序で指定してください。前の方のパラメータを省略するとき、後の方のパラメータを指定するのであれば、必要な数のカンマ (,) を指定しなければならないことに注意してください。*optlist2* のオプションを指定するときは、*optlist1* のオプションをすべて省略する場合でも、次のようにカンマが必要です。

```
OPEN "COM1: ,,,,CD1500" FOR INPUT AS #1
```

データビット長を 8 に設定すると、パリティに N (なし) を指定しなければなりません。Basic は、数値データに対して全バイト (8 ビット) を使うため、数値データの送受信を行うときは、データビット長を 8 に指定してください。

optlist2 引数には、*optlist1* 引数で指定した通信速度，パリティ，データ長，ストップビット以外の設定が指定できます。これらの設定はカンマで区切って指定します。どのような順番で指定してもかまいません。設定する内容は、データ モード，バッファサイズ，ハンドシェーキングの 3 種類に分けられます。表中の m はミリ秒 (1/1000 秒) 単位の意味です。

注　意　タイムアウト チェックを行わない場合、CS および DS には必ず 0 を指定してください。

データモード オプション (ASC, BIN, LF)

オプション	機能
ASC	ASCII モードでデバイスをオープンします。 ASCII モードでは、タブがスペースに展開され、行の終わりには強制的にキャリッジ リターン (CR) が置かれ、Ctrl+Z はファイルの終わり (EOF) として扱われます。通信チャネルをクローズすると、 RS-232C 回線を通じて、Ctrl+Z が送信されます。また、このモードでは XON/XOFF 制御を許可します。
BIN	バイナリモードでデバイスをオープンします (デフォルト)。バイナリモードでは、タブはスペースに展開されず、行の終わりに強制的にキャリッジ リターンが置かれません。また、 Ctrl+Z は、ファイルの終りとして扱われません。通信チャネルをクローズしても、 RS-232C 回線を通じて、Ctrl+Z が送信されません。
LF	ASC オプションを指定したときのみ有効です。通信ファイルをプリンタで印刷できるようにします。 LF オプションを指定すると、ライン フィード (0AH) が、自動的にキャリッジ リターン (0DH) の後ろに付け加えられます。行の幅を設定するために使ったキャリッジ リターンの後にもライン フィードが付け加えられます。 LF オプションを使ってオープンした通信ファイルからデータを読み込む **INPUT, LINE INPUT** ステートメントは、キャリッジ リターンまでのデータを読み込みますが、ライン フィードは無視します。

シーケンシャル モードのバッファサイズ オプション (RB)

オプション	機能
RB[*n*]	受信バッファのサイズを *n* バイトに設定します。*n* を省略したり、RB オプションそのものを省略すると、デフォルトの 512 バイトに設定されます。ただしこのデフォルトのサイズは、 BC または VBDOS の /C コマンドライン オプションで変更できます。バッファ サイズの最大値は 32767 バイトです。

ハンドシェークとタイミングのオプション (CD, CS, DS, OP)

オプション	機能
CD[*m*]	DCD (Data Carrier Detect) 回線上でのタイムアウトを制御します。 *m* ミリ秒が経過しても DCD がロー (Low: 論理 0) のままであると、デバイス タイムアウトが起こります。デフォルトの *m* の値は 0 です。0 が指定された場合は、タイムアウト チェックをしません。最大 65535 が指定できます。
CS[*m*]	CTS (Clear To Send) 回線上でのタイムアウトを制御します。 *m* ミリ秒が経過しても CTS がロー (Low: 論理 0) のままであると、デバイス タイムアウトが起こります。デフォルトの *m* の値は 1000 です。0 が指定された場合は、タイムアウト チェックをしません。最大 65535 が指定できます。特にタイムアウトを設定しない場合、必ず 0 を指定してください。
DS[*m*]	DSR (Data Set Ready) 回線上でのタイムアウトを制御します。 *m* ミリ秒を経過しても DSR がロー (Low: 論理 0) のままであると、デバイス タイムアウトが起こります。デフォルトの *m* の値は 1000 です。0 が指定された場合は、タイムアウトチェックをしません。最大 65535 が指定できます。特にタイムアウトを設定しない場合、必ず 0 を指定してください。

OP[*m*]	ファイルがオープンするまでの **OPEN COM** ステートメントの待ち時間を設定します。パラメータ *m* は、0～65,535 の値で、通信回線がアクティブになるまでの待ち時間をミリ秒単位で表します。 OP オプションでパラメータ *m* を省略すると、ファイルがオープンするまで 10 秒間待ちます。OP オプションを省略すると、CD および DS オプションで指定したタイムアウト値の 10 倍の時間待ちます。0 が指定された場合は、タイムアウト チェックをしません。通常は、0 を指定してください。指定する場合は、CS, DS, CD オプションに比べて大きな値を使ってください。

mode 引数には、次のいずれかのキーワードを指定してください。省略すると、**RANDOM** と見なされます。

モード	機能
OUTPUT	シーケンシャル出力モード
INPUT	シーケンシャル入力モード
RANDOM	ランダムアクセス モード

filenumber% 引数には、シリアルポートに割り当てるファイル番号を指定します。

reclen% 引数には、ランダムアクセス モードでオープンする場合に、ランダムアクセス バッファのレコード長を指定します。デフォルトの値は 128 です。**GET, PUT** などのランダムアクセス I/O ステートメントを使うと、オープンしたデバイスをランダムアクセス ファイルのように取り扱うことができます。

OPEN COM ステートメントは、次の手順で通信デバイスをオープンします。

1. 通信バッファを割り当て、割り込みをイネーブルにします。
2. DTR (Data Terminal Ready) 回線をハイ (High: 論理 1) に設定します。
3. OP オプションまたは DS オプションが 0 以外に設定されている場合は、DSR (Data Set Ready) 回線がハイ (High: 論理 1) になるまで、指定した時間待ちます。タイムアウトが起こると、処理は手順 6 に移ります。
4. RTS (Request To Send) をハイ (High: 論理 1) に設定します。
5. OP オプションまたは DS オプションが 0 以外に設定されている場合は、DCD (Data Carrier Detect) がハイ (High: 論理 1) になるまで、指定した時間待ちます。タイムアウトが起こると、処理は手順 6 に移ります。タイムアウトが起こらなければ、**OPEN COM** ステートメントの実行は終了します。

6. タイムアウトが起こったため、デバイスのオープンは失敗しました。バッファが解放され、割り込みもできなくなります。また、すべての制御回線がクリアされます。このとき、"デバイス タイム アウトです" というメッセージが表示されます。また、ERDEV$ 関数の値が COM に設定され、ERDEV 関数の値は、時間切れになったラインを示す次のような値に設定されます。

ERDEV 値	ライン
128 (80H)	CTS (Clear to Send) タイムアウト
129 (81H)	DSR (Data Set Ready) タイムアウト
130 (82H)	DCD (Data Carrier Detect) タイムアウト

OCOM.OBJ をリンクすると .EXE ファイルのサイズを縮小できます。

OPEN COM ステートメントに構文エラーがあると、"ファイル名が無効です" エラーが発生します。

参照　**OPEN** ステートメント (ファイル I/O)

OPTION BASE ステートメント

機能　配列の添字の、デフォルトの最小値を設定します。

構文　**OPTION BASE** {0|1}

解説　**DIM** または **REDIM** ステートメントで **TO** キーワードを使うと、より簡単で柔軟に、配列の添字の範囲を制御できます。通常は **OPTION BASE** ステートメントを使わずに、**DIM** または **REDIM** ステートメントで明確に添字の最小値を宣言するようにしてください。

DIM または **REDIM** ステートメントで、各次元の添字の最小値が省略されたときのデフォルトの値は 0 です。**OPTION BASE** ステートメントを使うと、このデフォルトの値を 1 に変更できます。

OPTION BASE ステートメントは、1 つのモジュールで 1 度だけ宣言できます。モジュール レベルの、すべての配列宣言より前に置かなければなりません。

参照　**LBOUND** 関数，**OPTION EXPLICIT** ステートメント，**UBOUND** 関数

OPTION EXPLICIT ステートメント

機能

すべての変数を宣言しなければ使えないようにします。

構文

OPTION EXPLICIT

解説

OPTION EXPLICIT ステートメントは、モジュール レベルの、すべての実行可能コードより前に置かなければなりません。通常はインクルード ファイル中に **OPTION EXPLICIT** ステートメントを記述して、そのファイルを **$INCLUDE** メタコマンドでインクルードするようにします。

OPTION EXPLICIT ステートメントを使うと、変数の暗黙の宣言がされなくなります。宣言をせずに変数を使おうとすると、実行時に“変数が宣言されていません”エラーが発生します。これによって、識別子の入力ミスなどをチェックすることができます。

変数の宣言は、**COMMON, DEF FN, DIM, FUNCTION, REDIM, SHARED, STATIC, SUB** のいずれかのステートメントを使って行います。

参照

COMMON ステートメント，**CONST** ステートメント，**DEF FN** ステートメント，**DIM** ステートメント，**FUNCTION** ステートメント，**REDIM** ステートメント，**SHARED** ステートメント，**STATIC** ステートメント，**OPTION BASE** ステートメント

OUT ステートメント

INP 関数 を参照してください。

Paint イベント

機能

オブジェクトの一部または全部が、画面上に表示されたときに発生します。

構文

SUB Form_Paint ()

SUB *ctlname*_**Paint** ([*Index* **AS INTEGER**])

Index　コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

フォーム，ピクチャー ボックス

解説

Paint イベントは、フォームまたはコントロールが移動，拡大，縮小されたり、上に重なって表示されていたウィンドウがなくなったときに発生します。また、**REFRESH** メソッドが実行されても発生します。

Paint イベントは、オブジェクトの AutoRedraw プロパティに深く関係しています。AutoRedraw プロパティが True (-1) に設定されているときは、必要に応じて自動的に再描画されるため、Paint イベントは発生しません。AutoRedraw プロパティが、デフォルト値である False (0) に設定されている場合は、そのオブジェクトの一部または全部が新たに画面上に見えるようになったときにも、内容は再表示されません。この場合、Paint イベントに対するイベント プロシージャを使って、新しく表示された領域に再描画を行ってください。

Paint イベントは、コード中から **PRINT** メソッドを使って出力するときにも役立ちます。この場合は、Paint イベントに対するイベント プロシージャを使って、すでに出力した内容を必要に応じて再描画します。

Paint イベントによって、イベントの処理がループしてしまう場合があります。つぎのような状況に対して発生する Paint イベントを利用することは避けてください。

- カレントのフォームまたはコントロールの移動およびサイズ変更
- サイズまたは外観に影響する変数の変更 (例えばオブジェクトの BackColor プロパティの変更など)
- カレントのフォームまたはコントロールに対する **REFRESH** メソッドの実行

これらの状況のいくつかには、Resize イベントがより適切です。ただし、意味があるのはオブジェクトを拡大するときだけです。また、Resize イベントに対するイベント プロシージャ中で **REFRESH** メソッドを使うと、フォームのサイズが変更されるたびに再描画が行われます。

参照

AutoRedraw プロパティ，**PAINT** ステートメント，**REFRESH** メソッド

PAINT ステートメント

機能

指定した色や塗りつぶしパターンを使って、領域を塗りつぶします。フォームが表示されている状態では利用できません。

構文

PAINT [**STEP**](*x!,y!*)[, [{*color%*|*tile$*}][, [*bordercolor%*]]]

STEP	座標を相対座標で指定することを示します。
(*x!,y!*)	塗りつぶしを始める点のビューポート座標。この点は、閉じた図形の内側または外側でなければなりません。内側なら、その図形の内部が塗りつぶされます。
color%	塗りつぶしに使う色を示すパレット番号。省略すると、グラフィックスのデフォルトの色と見なされます。
tile$	塗りつぶしの模様を表す文字列式。
bordercolor&	図形の境界線の色を示すパレット番号。領域の塗りつぶしは、この境界色が現れたところで終了します。省略すると、 *color%* と同じと見なされます。

解説

フォームが表示されているときに **PAINT** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **PAINT** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

ビューポート座標は、**WINDOW** ステートメントが実行されていなければ物理座標と同じです。物理座標は、**VIEW** ステートメントが実行されていなければ、画面左上角を原点とする絶対座標です。

STEP キーワードは、現在のグラフィックス出力位置 (直前にプロットされた点) からの相対座標であることを示します。例えば、グラフィックス出力位置が (10,10) のときに STEP (4, 5) と指定すると、座標 (10+4, 10+5) すなわち (14,15) を意味します。

塗りつぶしは、塗りつぶす色と同じ色の線に達したところで終了します。このため **PAINT** ステートメントでは、塗りつぶしの開始点 (*x!*, *y!*) に、画面やビューポートの外側にある座標を指定することもできます。

"タイリング" とは、文字列で表される塗りつぶしの模様のことです。この文字列は、1 文字 (8 ビット) で、横 8 ピクセル分の模様を表します。1 ピクセルの色を青 (B), 緑 (G), 赤 (R), 輝度 (I) の 4 枚のグラフィックス プレーンで表す構成の場合には、4 文字分 (32 ビット) で、色の付いた横 8 ピクセル分の模様が表せることになります。

模様を指定するには、まず ON のピクセルを 1, OFF のピクセルを 0 とした 2 進数を考えます。例えば、横方向に 1 ピクセルずつ OFF/ON を繰り返した模様は、01010101 で表されます。この値は、10 進数の 85, 16 進数の &H55 に当たります。この値を文字列に変換するためには、**CHR$** 関数を使います。したがって、明るい白 (すべてのプレーンが ON) でこの模様で塗りつぶすには、次のように指定します。

```
PAINT(x,y), CHR$(&H55) + CHR$(&H55) + CHR$(&H55) + CHR$(&H55)
```

CHR$(&H55)、つまり ASCII コードの &H55 に対応する文字は、"U" です。したがって次のようにも指定できます。この場合でも、文字 "U" は模様を示すだけで、実際に画面に表示されるわけではありません。

```
PAINT(x,y), "UUUU"
```

縦方向にも模様を付けたいときには、**CHR$** 関数 4 つが示す横 8 ピクセルずつの模様を、連結して指定します。次のような構文になります。

```
TILE$ = CHR$(arg1B) + CHR$(arg1G) + CHR$(arg1R) + CHR$(arg1I)
TILE$ = TILE$ + CHR$(arg2B) + CHR$(arg2G) + CHR$(arg2R) +
CHR$(arg2I)
.
.
.
TILE$ = TILE$ + CHR$(argnB) + CHR$(argnG) + CHR$(argnR) +
CHR$(argnI)
PAINT(x,y), TILE$
```

塗りつぶす領域が、指定した模様のサイズより大きければ、模様は縦にも横にも繰り返して使われます。

参照　**CHR$** 関数，Paint イベント

PALETTE, PALETTE USING ステートメント

機能　パレットの色を設定します。フォームが表示されている状態では利用できません。

構文

PALETTE[*paletteno%*, *colorvalue%*]

PALETTE USING *arrayname*[(*arrayindex%*)]

paletteno%	設定を変更するパレット番号を示す整数。
colorvalue%	*paletteno%* で指定したパレット番号に割り当てる色を示す色値。
arrayname	それぞれのパレット番号に割り当てる色値が格納された配列変数の名前。
arrayindex%	パレット番号 0 に対する色値が格納されている要素の添字。省略すると先頭の要素と見なされます。通常は省略します。

解説　フォームが表示されているときに **PALETTE** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

PALETTE ステートメントは、グラフィックスの色を指定するときに使うパレットの設定を変更します。通常の **PALLETE** ステートメントは一度に 1 つのパレット番号に対して変更を行うのに対し、**USING** キーワードを指定した **PALETTE** ステートメントは一度にパレット全体の設定を変更します。

パレット番号の最大値やパレットのデフォルトの設定、色値として有効な値の範囲は、使用しているコンピュータの機種や **SCREEN** ステートメントの指定によって決まります。「機種依存情報」を参照してください。

USING キーワードを指定しない場合の **PALETTE** ステートメントは、*paletteno%* で指定したパレット番号に対して、*colorvalue%* で指定した色値を割り当てます。以降、図形を描画するステートメントでパレット番号 *paletteno%* を指定すると、新しく設定した色値 *colorvalue%* が示す色で図形が描かれます。また、そのときすでに画面上に表示されている、パレット番号 *paletteno%* で描かれた図形の色は、即座に *colorvalue%* が示す色に変わります。

引数を省略して **PALETTE** ステートメントを実行すると、パレットはそのときのスクリーン モードのデフォルト値の設定になります。

例えば 4096 色中から 16 色表示できる設定の場合に次のステートメントを実行すると、パレット番号 5 にはグレーが割り当てられます。

```
PALETTE 5, &H777
```

USING キーワードを指定した **PALETTE** ステートメントは、一度でパレット全体の設定を変更します。*arrayname* 引数には、あらかじめ必要な数の色値を格納しておいた配列変数の名前を指定します。配列変数の先頭の要素に格納されている色値から順に、パレット番号に割り当てられます。

```
DIM Pal%(0 TO 7)
.
.
.
PALETTE USING Pal%
```

この例では、Pal%(0) に格納されている色値がパレット番号 0, Pal%(1) がパレット番号 1 というように割り当てられます。配列変数の要素数が、パレット番号の数 (同時表示可能な色数) より少ないときはエラーになります。

配列変数の途中の要素から値を使うようにしたいときは、*arrayindex* 引数に使用する先頭の要素の添字を指定します。

```
DIM Pal%(0 TO 100)
.
.
.
PALETTE USING Pal%(5)
```

この例で、パレット番号の最大値が 15 (16 色表示) なら、Pal%(5) ～ Pal%(20) までの要素が使われます。

配列変数の要素に -1 を格納しておくと、対応するパレット番号の設定を変更しないという意味になります。-1 以外の負の値を指定するとエラーになります。

グラフィックスが表示できるスクリーン モードでは、パレット番号 0 に設定されている色が画面全体の背景色になります。

グラフィックスのデフォルトの色を設定するには、**COLOR** ステートメントを使ってください。

参照　**SCREEN** ステートメント，**COLOR** ステートメント

Parent プロパティ

機能　コントロールや子フォームの親に当たるフォームを示します。デザイン時には使用できず、実行時には読み込み専用です。

構文　[*formname.*][*ctlname.*]**Parent**

対象　チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，テキスト ボックス，タイマー，垂直スクロール バー

解説　Parent プロパティは、コンテナ フォームのプロパティ，メソッド，コントロールなどにアクセスするために使います。例えば、コントロールを引数としてジェネラル プロシージャに引き渡す場合、ジェネラル プロシージャ中では、渡されたコントロールを元にその親に当たるフォームに対して操作することができます。

```
MyButton.Parent.MousePointer = 4
```

フレームやピクチャー ボックスの子コントロールの Parent プロパティも、フレームやピクチャー ボックスではなくフォームを示します。

Parent プロパティは、デザイン時には使用できず、フォーム デザイナの "ﾌﾟﾛﾊﾟﾃｨ" 欄にも表示されません。また、実行時は読み込み専用です。

Parent プロパティの値を直接参照することはできません。コンテナ フォームのプロパティを参照したり、コンテナ フォームに対してメソッドを使うためだけに利用できます。

Path プロパティ

機能

ディレクトリ リスト ボックスまたはファイル リスト ボックスでそのとき展開されているディレクトリ パスの取得と設定を行います。デザイン時には無効です。

構文

[*formname.*]*ctlname*.**Path**[= *pathstring$*]

ctlname	ディレクトリ リスト ボックスまたはファイル リスト ボックスの名前。
pathstring$	ディレクトリを示す文字列式。そのとき有効なディレクトリ名でなければなりません。デフォルトは、コントロールがロードされた時点でのカレント ディレクトリです。

対象

ディレクトリ リスト ボックス，ファイル リスト ボックス

解説

pathstring$ の指定方法は、MS-DOS の chdir コマンドと同じです。つまり、相対パスでも指定可能であり、ドライブ名は省略できます。ドライブ名だけを指定すると、そのドライブのデフォルトのディレクトリが選択されます。

ディレクトリ リスト ボックスの Path プロパティの設定を変更すると、Change イベントが発生します。また、ファイル リスト ボックスの Path プロパティの設定を変更すると、PathChange イベントが発生します。

ディレクトリ リスト ボックスでは、Path プロパティと List および ListIndex プロパティでは意味が異なることに注意してください。List および ListIndex プロパティでは反転表示されている項目が対象であるのに対し、Path プロパティは表示の中心となっているディレクトリ (展開されているディレクトリ) が対象です。

```
ctlname.Path
ctlname.List(ctlname.ListIndex)
```

参照

Archive プロパティ，Change イベント，ドライブ リスト ボックス，Drive プロパティ，FileName プロパティ，Hidden プロパティ，List プロパティ，ListCount プロパティ，ListIndex プロパティ，Normal プロパティ，PathChange イベント，Pattern プロパティ，ReadOnly プロパティ，System プロパティ

PathChange イベント

機能

コード中から、FileName または Path プロパティが設定され、パスが変更されたときに発生します。

構文

SUB *ctlname*_**PathChange** ([*Index* **AS INTEGER**])

Index コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

ファイル リスト ボックス

解説

PathChange イベントは、ファイル リスト ボックスのディレクトリ パスが変更されたとき、つまり新しいパスを含む文字列が FileName プロパティに代入されるか、新しいパスを示す文字列が Path プロパティに代入されると発生します。

参照

FileName プロパティ，Path プロパティ，Pattern プロパティ，PatternChange イベント

Pattern プロパティ

機能

ファイル リスト ボックスに表示されるファイル名を示すパターンを決めます。

構文

[*formname*.]*ctlname*.**Pattern**[= *filespec$*]

ctlname ファイル リスト ボックスの名前。

filespec$ 表示するファイルのパターンを示す文字列式。ワイルドカード (? または *) が使えます。デフォルトは "*.*" です。

対象

ファイル リスト ボックス

解説

Pattern プロパティは、特定の種類のファイルだけをファイル リスト ボックスに表示したいときに使います。例えば、実行可能ファイルだけをファイル リスト ボックスに表示したいとき、パターンは *.EXE になります。オペレーティング システムでファイルを指定するときと同様に、? は任意の 1 文字，* は任意の数の任意の文字を表します。

複数のパターンを組み合わせて指定したいときは、セミコロンで区切ります。例えば拡張子 .FRM と .BAS のファイルだけを表示したいときは、"*.FRM ; *.BAS" を Pattern プロパティに代入します。

Pattern プロパティの値が変更されると PatternChange イベントが発生します。

参照　Archive プロパティ，Drive プロパティ，FileName プロパティ，Hidden プロパティ，Normal プロパティ，Path プロパティ，PatternChange イベント，ReadOnly プロパティ，System プロパティ

PatternChange イベント

機能　コード中から、FileName または Pattern プロパティが設定され、パターンが変更されたときに発生します。

構文　**SUB** *ctlname*_**PatternChange** ([*Index* **AS INTEGER**])

Index　コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象　ファイル リスト ボックス

解説　PatternChange イベントは、ファイル リスト ボックスでファイルを指定するパターン (ワイルド カードを含む文字列) が変更されたとき、つまり新しいパターンを含む文字列が FileName プロパティに代入されるか、新しいパターンを示す文字列が Pattern プロパティに代入されると発生します。

参照　FileName プロパティ，Path プロパティ，PathChange イベント，Pattern プロパティ

PCOPY ステートメント

機能　あるグラフィックス スクリーン ページを、別のページにコピーします。フォームが表示されている状態では利用できません。

構文　**PCOPY** *sourcepage%, destinationpage%*

sourcepage%　コピー元のグラフィックス スクリーン ページの番号を示す 0～*n* の整数。

destinationpage%　コピー先のグラフィックス スクリーン ページの番号を示す 0～*n* の整数。

解説　フォームが表示されているときに **PCOPY** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **PCOPY** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

スクリーン ページの枚数は、使用している機種や、そのときのスクリーン モードによって決まります。引数に指定できる最大値 *n* は、この状態によって異なります。最初のページのページ番号は 0 です。

各スクリーン モードで使用可能なページ数は、**SCREEN** ステートメントの *apage%* および *vpage%* 引数に指定できるものと同じです。

参照　**SCREEN** ステートメント

PEEK 関数, POKE ステートメント

機能　**PEEK** 関数は、メモリ上の指定した位置に格納されているバイト データを返すメモリ管理関数です。**POKE** ステートメントは、メモリ上の指定した位置にバイト データを書き込むメモリ管理関数です。

構文

PEEK(*address*)

POKE *address*, *byte%*

address	直前の **DEF SEG** ステートメントで指定したカレント セグメントに対するオフセットを表す、0～65,535 の値。
byte%	メモリに書き込むデータを表す、0～255 の整数。

解説　**POKE** ステートメントを使って文字列を直接操作するときには注意が必要です。使い方を誤ると、Visual Basic やオペレーティング システムが誤動作する場合があります。

Basic は、プログラム実行中に文字列の格納位置を移動することがあります。**POKE** ステートメントや **PEEK** 関数の直前で、**DEF SEG** ステートメントを実行するようにしてください。

address 引数に浮動小数点数や長整数型を指定すると、整数 (2 バイト) に変換されます。

PEEK 関数と **POKE** ステートメントはそれぞれ対になる機能を持っています。

PEEK 関数で far 文字列配列から 1 バイトを取り出す場合は、**SSEG** と **SADD** 関数を使ってカレントのセグメントおよびオフセット アドレスを調べます。

```
DEF SEG = SSEG(a$)          ' カレントのセグメント アドレスを
                            ' a$ のアドレスに設定します。
StringOffset = SADD(a$)     ' セグメント中での文字列の位置を
                            ' 調べます。
PEEK(StringOffset)          ' このアドレスに保存されている
                            ' 1 バイトを得ます。
```

POKE ステートメントで far メモリ中に保存されているデータを操作する場合は、**DEF SEG** ステートメントと **SSEG** 関数を使ってカレントのセグメント アドレスを設定します。

```
DEF SEG = SSEG(a$)              ' カレントのセグメント アドレスを
                                ' a$ のアドレスに設定します。
Offset1 = SADD(a$)              ' セグメント中での文字列の位置を
                                ' 調べます。
POKE Offset1, PEEK(Offset2)     ' 1 バイト書き込みます。
```

PEEK 関数や **POKE** ステートメントは、EMS メモリ内の配列には使わないでください。/Ea オプションを指定して Visual Basic (VBDOS.EXE) を起動した場合、次の配列は EMS メモリに格納されることがあります。

- 512 バイト～ 16K バイトの数値配列
- 512 バイト～ 16K バイトの固定長文字配列
- 512 バイト～ 16K バイトのユーザー定義型配列

配列に対して **PEEK** 関数や **POKE** ステートメントを使いたいときは、/Ea オプションを指定せずに Visual Basic (VBDOS.EXE) を起動してください。この場合、配列は EMS メモリには格納されません。

参照 **DEF SEG** ステートメント

PLAY ステートメント (イベント トラッピング)

機能 ミュージック イベントのトラップを有効，無効，中断状態にします。フォームやコントロールに関連したイベントには影響を与えません。

構文 **PLAY {ON|OFF|STOP}**

ON トラップを有効にすることを示します。 **ON PLAY** ステートメントによる、ミュージック イベントのトラップが可能になります。

OFF トラップを無効にすることを示します。このステートメントが実行されたあとに起こったイベントは、記憶されません。

STOP トラップを中断します。このステートメントが実行されたあとでミュージック イベントが発生すると、そのイベントは記憶されます。次に **PLAY ON** ステートメントが実行されると、すぐに **ON PLAY** ステートメントでトラップされます。

解説 このステートメントで扱うイベントは、フォームやコントロールを使ったイベント ドリブン プログラミング モデルのイベントとは異なることに注意してください。フォームやコントロールに関連したイベントには影響を与えません。ただし、フォームやコントロールを使ったアプリケーションでも使うことができます。

イベント トラップが有効であるとき、そのイベントが発生していないかどうか (キューの中にある、バックグラウンドで演奏される音符の残りが設定されている数より少なくなっていないか) が自動的に監視されています。このチェックは、プログラミング環境で実行されている場合と、/V オプション付きでコンパイルされた実行可能ファイルでは、ステートメントごとに行われます。/W オプション付きでコンパイルされた実行

可能ファイルでは、行番号または行ラベルごとにチェックされます。イベントが発生していると、**ON PLAY** ステートメントで設定されているイベント処理ルーチンに制御が移ります。

イベント トラップを行うステートメントを含むプログラムを BC コマンドラインからコンパイルするとき、/W または /V オプションを付けずにコンパイルすると、"ON event ステートメントを含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

PLAY STOP ステートメントでトラップを中断している間に起きたイベントは、**PLAY OFF** ステートメントが実行されると失われます。

イベントがトラップされ、対応する処理ルーチンに制御が移ると、再帰的なトラップが起こらないように自動的に **PLAY STOP** が実行されます。処理ルーチンが終了すると、そのルーチンの中で明確に **PLAY OFF** が実行されていない限り、自動的に **PLAY ON** ステートメントが実行されます。

参照　**ON PLAY** ステートメント，**PLAY** 関数，**PLAY** ステートメント (ミュージック)

PLAY ステートメント (ミュージック)

機能　音楽を演奏します。

構文　**PLAY** *commandstring$*

commandstring$　演奏するミュージック コマンドを含む文字列。

解説　*commandstring$* には、1 つ以上の音楽マクロ コマンドを含む文字列式を指定します。**PLAY** ステートメントは、**DRAW** ステートメントと同じように、1 つのステートメントの中にマクロコマンドを組み込んで実行します。**PLAY** ステートメントの一部として使う各マクロ コマンドは、それぞれ特定の動作を指定します。

オクターブと音の高さを指定するコマンド

オクターブ	動作
O*n*	現在のオクターブを設定します。*n* には 0～6 の整数を指定し、7 オクターブを選択します。
>	1 オクターブ上げます。最高は 6 オクターブ (*n*=6) です。
<	1 オクターブ下げます。最低は 0 オクターブ (*n*=0) です。

音の高さ	動作
A～G	演奏する、A～G の音符を指定します。音符の後ろに "#" か "+" を置くと、シャープが指定できます。また、音符の後ろに "-" を置くと、フラットを指定できます。
N*n*	音の高さ *n* の音符を演奏します。*n* の範囲は 0～84 です(0～84 で 7 オクターブの音階を表現します)。

サフィックス	動作
#, +	指定した音符の後ろに置いて、半音上げます (シャープ)。
-	指定した音符の後ろに置いて、半音下げます (フラット)。

音の長さとテンポを指定するコマンド

音の長さ	動作
L*n*	各音符の長さを設定します。L4 は四分音符を表し、L1 は全音符を表します。*n* の範囲は 1～64 です。特定の音符の長さだけを変えたいときは 、音符の後ろに *n* の値だけを指定してもかまいません。例えば、 A16 は L16 A と同じ意味です。
MN	"ノーマル" を設定します。各音符は、長さ (L) で指定した時間の 7/8 の長さで演奏されます。
ML	"レガート" を設定します。各音符は、長さ (L) で指定した完全な長さで演奏されます。
MS	"スタッカート" を設定します。各音符は、長さ (L) で指定した時間の 3/4 の長さで演奏されます。

テンポ	動作
P*n*	休符の長さを指定します。*n* の範囲は 1～64 で、意味は L*n* の *n* と同じです。
T*n*	"テンポ" または 1 分間に演奏する四分音符の数を設定します。*n* の範囲は 32～255 で、デフォルトは 120 です。 注意: 割り込み間隔が長い場合、音符によってはテンポが遅くなります (例えば、T255 での L64)。 音符の後ろにピリオドを置くと、 L(長さ), T(テンポ) で設定した 3/2 倍の長さで音符を演奏します。ピリオドは、音楽記号の付点と同じ意味です。音符の後ろにいくつでもピリオドを置くことができます。複数のピリオドを指定すると、各ピリオドは、その前にあるピリオドの 1/2 の長さを付け加えます。 例えば "A." は、指定した長さの 1+1/2 または 3/2 倍の時間、音符 A を演奏します。 "A.." は、指定した長さの 1+1/2+1/4 または 7/4 倍の時間、音符 A を演奏します。 ピリオドは、休符 (P) の後ろにも置くことができます。この場合、休止の長さは、音符と同じ規則で長くなります。

フォアグラウンド/バックグラウンドでの演奏方法

演奏方法	動作
MF	音楽 (**PLAY** ステートメント) と **SOUND** ステートメントをフォアグラウンドで実行するよう設定します。つまり、このコマンドに続く音符と音の演奏は、その前に置かれた音符と音の演奏が終わるまで、始まりません。これはデフォルトです。
MB	音楽 (**PLAY** ステートメント) と **SOUND** ステートメントをバックグラウンドで実行するよう設定します。つまり、音符や音はバッファに読み込まれ、バックグラウンドでこれらの音符や音を演奏しながら、 Basic のプログラムを実行することができます。最大 32 個の音符を 一度にバックグラウンドで演奏することができます。

X コマンド

マクロ文字列	動作
"X" + **VARPTR$**(*string*)	別のマクロ文字列を実行します。

X コマンドは、上記のコマンド文字を組み合わせた文字列をマクロのように取り扱い、あるマクロ文字列から別のマクロ文字列を実行するために使います。1 つのマクロ文字列から別のマクロ文字列を実行し、このマクロ文字列から、さらに別のマクロ文字列を実行するというように、次々にマクロ文字列を実行することができます。マクロ文字列を指定するには、必ず **VARPTR$** 関数を使います。

PLAY "X"+ **VARPTR$**(*commandstring$*)

C-, B+, E+, F- を指定すると "引数が有効範囲にありません" エラーになります。

参照　**BEEP** ステートメント, **PLAY** ステートメント (イベント トラッピング), **SOUND** ステートメント

PLAY 関数

機能　バックグラウンド ミュージック用のキューの中に現在ある音符の数を返します。

構文　**PLAY**(*n*)

n　ダミーの引数で、どんな数値でもかまいません。

解説　フォアグラウンドで演奏されている場合は、0 を返します。

参照　**ON PLAY** ステートメント, **PLAY** ステートメント (イベント トラッピング), **PLAY** ステートメント (ミュージック)

PMAP 関数

機能

ビューポート座標を物理座標に、物理座標をビューポート座標に変換して返します。フォームが表示されている状態では利用できません。

構文

PMAP (*startcoordinate#, n%*)

startcoordinate#	変換元の座標を示す数式。
n%	変換方法を示す、0～3 の整数。

解説

フォームが表示されているときに **PMAP** 関数を参照すると、"フォームが表示されているときは無効です" エラーが発生します

引数 *n%* には、変換方法を示す次のいずれかの値を指定します。

値	意味
0	指定した *startcoordinate#* がビューポート座標系の x 座標を表すことを示します。物理座標系の x 座標に変換します。
1	指定した *startcoordinate#* がビューポート座標系の y 座標を表すことを示します。物理座標系の y 座標に変換します。
2	指定した *startcoordinate#* が物理座標系の x 座標を表すことを示します。ビューポート座標系の x 座標に変換します。
3	指定した *startcoordinate#* が物理座標系の y 座標を表すことを示します。ビューポート座標系の y 座標に変換します。

PMAP ステートメントを使うと、1 つの点の座標を、**WINDOW** ステートメントで作成したビューポート座標系と、画面の物理座標系や **VIEW** ステートメントで定義したビューポートの物理座標系の両方で表すことができます。

参照

POINT 関数，**VIEW** ステートメント，**WINDOW** ステートメント

POINT 関数

機能

指定した座標の現在の色、または現在のグラフィックス出力位置の座標を返すグラフィックス関数です。フォームが表示されている状態では利用できません。

構文

POINT {(*number%*)|(*x!,y!*)}

number%	現在のグラフィックス出力位置に関して調べたい項目の種類を示す、0～3 の整数。
(*x!,y!*)	パレット番号を調べたいピクセルのビューポート座標。指定した座標がビューポートの外側のときは -1 を返します。

解説

フォームが表示されているときに **POINT** 関数を参照すると、"フォームが表示されているときは無効です" エラーが発生します。

(*x!*,*y!*) を指定すると、その座標のピクセルの、現在の色を示すパレット番号を得ることができます。指定した座標が画面の外にあるときは、-1 を返します。

number% 引数を指定すると、現在のグラフィックス出力位置 (次のグラフィックス ステートメントの描き始めの位置) の座標を得ることができます。*number%* には、調べたい項目を示す、次のいずれかの整数を指定します。

引数	戻り値
0	x 方向の位置を物理座標で返します。
1	y 方向の位置を物理座標で返します。
2	x 方向の位置をビューポート座標で返します。 **WINDOW** ステートメントで領域が設定されていなければ、 0 と同じ意味です。
3	y 方向の位置をビューポート座標で返します。 **WINDOW** ステートメントで領域が設定されていなければ、 1 と同じ意味です。

参照

COLOR ステートメント, **PMAP** 関数, **SCREEN** ステートメント, **VIEW** ステートメント, **WINDOW** ステートメント

POKE ステートメント

PEEK 関数を参照してください。

POS 関数, CSRLIN 関数, LOCATE ステートメント

機能

POS 関数は、現在のカーソルの桁位置を返し、**CSRLIN** 関数は行位置を返します。**LOCATE** ステートメントは、指定した位置にカーソルを移動します。フォームが表示されている状態では利用できません。

構文

POS(*n%*)

CSRLIN

LOCATE [*row%*] [,[*column%*] [,[*cursor%*] [,*start%* [,*stop%*]]]]

n% 任意の数式。この引数は、 **POS** が関数であることを示すだけであり、値に意味はありません。

row% カーソルの新しい行位置を示す整数。省略すると行位置は変わりません。

column% カーソルの新しい桁位置を示す整数。省略すると桁位置は変わりません。

cursor% カーソルを表示するかどうかを示す値。0 を指定すると表示されず、1 を指定すると表示されます。省略すると、元の状態のままです。ただし非表示にしても、**INPUT** ステートメントなどを実行したときはカーソルは表示されます。また、グラフィックスモードではカーソルは表示されません。

start% カーソルの形状を変更するときに、カーソルの上端の位置を示す整数。省略すると形状は変わりません。

stop% カーソルの形状を変更するときに、カーソルの下端の位置を示す整数。省略すると、*start%* と同じと見なされます。*start%* を省略したときは *stop%* も指定できません。

解説

フォームが表示されているときに **POS** および **CSRLIN** 関数を参照するか **LOCATE** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

カーソルが画面左端の列にあるとき、**POS** 関数は 1 を返します。カーソルが画面上端の行にあるとき、**CSRLIN** 関数は 1 を返します。

LOCATE ステートメントの引数はいずれも省略することができます。ただし *stop%* を指定するときは、同時に *start%* も指定します。指定されている引数にしたがって、カーソルの位置の移動，表示の ON/OFF，形状の変更を行います。

カーソルの上端，下端の位置として指定できる値の範囲は、スクリーンモードに依存します。

画面の最下行は、ファンクションキーに割り当てられている文字列を表示するために確保されており、ここにカーソルを置くことはできません。ただし、この表示をオフにし (KEY OFF), **LOCATE** ステートメントと **PRINT** ステートメントを組み合せて使うと、最下行にテキストを表示することができます。この場合、**PRINT** ステートメントの最後にはセミコロン (;) を指定してください。

参照

SCREEN ステートメント

PRESET, PSET ステートメント

機能

画面上の指定した座標に点を描きます。フォームが表示されている状態では利用できません。

構文

PRESET [**STEP**](*x!,y!*)[, *color%*]

PSET [**STEP**](*x!,y!*)[, *color%*]

STEP	座標を相対座標で指定することを示します。
(*x!,y!*)	点を描く位置を示すビューポート座標。
color%	点の色を表すパレット番号。省略すると、 **PRESET** ステートメントではパレット番号 0 (画面全体の背景色) と見なされ、**PSET** ステートメントではグラフィックスのデフォルトの色と見なされます。

解説

フォームが表示されているときに **PRESET** または **PSET** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **PRESET** または **PSET** ステートメントを実行すると、"引数が有効範囲にありません" エラーが発生します。

ビューポート座標は、**WINDOW** ステートメントが実行されていなければ物理座標と同じです。物理座標は、**VIEW** ステートメントが実行されていなければ、画面左上角を原点とする絶対座標です。

STEP キーワードは、現在のグラフィックス出力位置 (直前にプロットされた点) からの相対座標であることを示します。例えば、グラフィックス出力位置が (10,10) のときに STEP (4, 5) と指定すると、座標 (10+4, 10+5) すなわち (14,15) を意味します。

PRESET と **PSET** ステートメントは、*color%* 引数を省略したときのデフォルトを除き、機能は同じです。

現在のビューポートの外側の座標を指定しても何も行われません。このときエラーにはなりません。

参照

SCREEN ステートメント，**VIEW** ステートメント，**WINDOW** ステートメント

PRINT メソッド

機能

フォーム，ピクチャー ボックス，または **PRINTER** オブジェクトに、数値または文字列を出力します。指定した書式に従って出力することもできます。

構文

[{{*formname*.]*ctlname*.|*formname*.|**PRINTER**.}]**PRINT** [*expression*[{,|;}*expression*]...[{,|;}]]

[{{*formname*.]*ctlname*.|*formname*.|**PRINTER**.}]**PRINT USING** *formatstr$*;
expression[; *expression*...][;]

ctlname	ピクチャー ボックスの名前。
formatstr$	出力するデータの書式を示す文字列式。
expression	出力するデータを示す数式または文字列式。省略すると、単に改行されます。複数の式を指定するときは、セミコロン (;) またはカンマ (,) で区切ります。
;	データを出力したあと、出力位置を進めないことを示します。次のデータは、この直後から出力されます。 **USING** キーワードを指定した場合、*expression* の間のセミコロンは単なる区切りで、最後のセミコロンだけが意味を持ちます。
,	データを出力したあと、次の出力領域 (14 桁ごと) に出力位置を進めることを示します。次のデータは、次の出力領域の先頭から出力されます。

対象

フォーム，ピクチャー ボックス，**PRINTER** オブジェクト

解説

PRINT メソッドは **PRINT** ステートメントに似ていますが、**PRINT** メソッドの出力はオブジェクトに送られるという点が異なります。

フォーム モジュールで、(対象のオブジェクトを指定せずに) **PRINT** キーワードを単独で使うと、フォームに対する **CLS** メソッドと解釈されます。これに対してコード モジュール中では、**CLS** ステートメントと解釈されます。

PRINTER オブジェクトに対して適用するとき、出力は **PRINTER** オブジェクトの PrintTarget プロパティに設定されているプリンタまたはファイルに送られます。

出力は、そのときの CurrentX および CurrentY プロパティで示される位置から開始されます。*expression* 引数の最後のセミコロンまたはカンマの意味は、**PRINT** ステートメントと同じです。つまり最後がセミコロンなら、CurrentX および CurrentY プロパティの値は変わりません。カンマなら、CurrentX プロパティに次の出力領域 (14 桁ごと) の先頭を示す値が設定されます。最後に何も付けなければ、CurrentY に 1 が加えられ、CurrentX は 0 になります。

出力されたデータがフォームやピクチャー ボックスに収まらないときでも、自動スクロールは行われません。フォームやピクチャー ボックスが水平および垂直スクロールバーを持っていれば、スクロールすると、境界を超えて出力されたデータが表示されます。

出力されたデータの長さがプリンタの行幅を超えるときは、切り捨てられるか改行されるかは、プリンタの設定によって決まります。ただし改ページは自動的に行われます。

プリンタに対して強制的に改ページしたいときは、**NEWPAGE** メソッドを使います。

USING キーワードで指定する書式を指定する文字列は、**PRINT** ステートメントと同じです。

SPC や **TAB** などの関数を使って、出力位置を制御することができます。

参照　CurrentX プロパティ，CurrentY プロパティ，**LPRINT** ステートメント，**PRINT** ステートメント，**PRINT #** ステートメント，**PRINTFORM** メソッド，**TAB** 関数，**SPC** 関数

PRINT ステートメント

機能

数値または文字列データを画面に出力します。指定した書式に従って出力することもできます。フォームが表示されている状態では利用できません。また、フォーム モジュール中では使用できません。**PRINT** メソッドを使ってください。

構文

PRINT [*expression*[{,|;}*expression*]...[{,|;}]]

PRINT USING *formatstr$*; *expression*[; *expression*]...[;]

expression	出力するデータを示す数式または文字列式。省略すると、単に改行されます。複数の式を指定するときは、セミコロン (;) またはカンマ (,) で区切ります。
;	データを出力したあと、出力位置を進めないことを示します。次のデータは、この直後から出力されます。 **USING** キーワードを指定した場合、*expression* の間のセミコロンは単なる区切りで、最後のセミコロンだけが意味を持ちます。
,	データを出力したあと、次の出力領域 (14 桁ごと) に出力位置を進めることを示します。次のデータは、次の出力領域の先頭から出力されます。
formatstring$	出力するデータの書式を示す文字列式。

解説

PRINT がステートメントとして解釈されるか **PRINT** メソッドと解釈されるかは、キーワード **PRINT** がコード中のどの位置で使われているかによって決まります。フォーム モジュールでは、**PRINT** キーワードは **PRINT** ステートメントではなく **PRINT** メソッドと解釈されます。**PRINT** キーワードがフォーム モジュール中のジェネラル プロシージャ中にあっても同じです。つまり、**PRINT** がステートメントとして解釈されるのは、コード モジュール中で、オブジェクトを指定せずに単独で **PRINT** と記述されている場合だけです。

フォーム モジュールから **PRINT** ステートメント特有の機能を利用するには、**PRINT** ステートメントをコード モジュール中のジェネラル プロシージャに入れて、それを呼び出すようにします。

```
SUB Form_Click                    'フォーム モジュール
    SCREEN.HIDE
    CALL MyPrint("Hello")
    SCREEN.SHOW
END SUB
SUB MyPrint (temp$)               'コード モジュール
    PRINT temp$
END SUB
```

フォームが表示されているときに **PRINT** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

PRINT ステートメントで数値データを出力すると、その後ろには自動的にスペースが 1 つ入れられます。また、数値が正の値なら、数値の前にもスペースが 1 つ入れられます。数値が負の値なら、数値の前にマイナス記号 (-) が付けられます。

PRINT ステートメントで単精度および倍精度の数値を表示する場合、固定小数点表記で表示される場合と浮動小数点表記で表示される場合があります。単精度の数値を、精度を落とさずに 7 桁以下の固定小数点で表すことができれば、固定小数点表記を使います。そうでなければ、浮動小数点表記になります。例えば、1.1E-6 という数値は .0000011 と表示され、1.1E-7 は 1.1E-7 と表示されます。同様に、倍精度の数値を精度を落とさずに 15 桁以下の固定小数点で表すことができれば、固定小数点表記を使います。そうでなければ、浮動小数点表記になります。例えば、1.1D-14という数値は .000000000000011 と表示され、1.1D-15 は 1.1D-15 と表示されます。

PRINT および **PRINT USING** ステートメントは、Basic の基本データ型 (整数，長整数，単精度浮動小数点数，倍精度浮動小数点数，通貨型，文字列) だけをサポートします。ユーザー定義型のデータを表示したいときは、次のように各要素ごとに指定してください。

```
TYPE MyType
    Word AS STRING * 20
    Count AS LONG
END TYPE
DIM Myrec AS MyType
PRINT Myrec.Word
```

表示位置

PRINT ステートメントでは、それぞれのデータ項目を表示する位置は、指定された区切り記号 (セミコロンまたはコロン) によって決まります。画面上の 1 行は、14 桁ごとの出力領域に分けられています。直前のデータの後ろにカンマ (,) が指定されていれば、次のデータは次の出力領域の先頭から出力されます。直前のデータの後ろにセミコロン (;) が指定されていれば、次のデータは前のデータのすぐあとから出力されます。

同様に、次の出力ステートメントの表示開始位置も、**PRINT** ステートメントの最後に指定された区切り記号によって決まります。最後にセミコロンおよびコロンを指定しなければ、次の出力ステートメントは、次の行の先頭から表示を始めます。したがっ

て、引数をすべて省略すると、改行だけを行います。また、表示する行の長さが画面の幅を超える場合、次の行に折り返して表示が続けられます。

プログラミング環境では、*expression* の間の区切り記号の代わりにスペースやタブを入力すると、セミコロンに置き換えられます。

書式指定

USING キーワードを指定すると、書式を指定してデータを画面に出力することができます。*formatstring$* には、文字列データや数値データの書式を制御するための文字列式を指定します。この中に含まれる書式指定文字以外の部分は、リテラル文字列としてそのまま出力されます。1 つの *formatstring$* で、複数の *expression* に対する書式を指定することもできます。指定した *expression* の数が *formatstring$* 中の書式の数より多ければ、*formatstring$* は繰り返し使われます。

文字列データの書式指定に使う文字

文字	規則
!	文字列の最初の文字だけを表示します。ただし、先頭の文字が 2 バイト系文字なら、スペースが表示されます。
@	文字列全体を表示します。
& &	2 つのアンパサンド (&) の間のスペースの個数に、2 を加えた数の文字を表示します。対応する文字列データの長さがこれより長い場合、残りの部分は表示されません。

数値データの書式指定に使う文字

文字	規則
#	1 つを 1 桁として、桁位置と桁数を指定します。それぞれの桁位置はつねに埋められます。もし指定された位置よりも出力される数字の桁数が少ない場合は、指定された桁数に対して右詰めで表示され、先頭はスペースで埋められます。
.	小数点の位置を指定します。小数点の前または後ろ、あるいは両方には、必ず桁位置 (#) がなければなりません。小数点以上の桁位置が指定されており、対応する数値データが小数点以下の値であれば、1 の位は 0 で埋められ、それ以上の桁はスペースで埋められます。
,	小数点以上 3 桁ごとに置き、桁区切りのカンマを表示します。小数点以上 3 桁ごと以外の位置に指定されると、リテラル文字列と解釈されることがあります。指数形式の場合に指定しても無視されます。
+	正負の符号 (+ か -) を表示します。書式設定文字列の最初に指定すれば、符号は数値データの前に表示され、書式設定文字列の最後に指定すれば、後ろに表示されます。
-	書式指定文字列の最後に指定すると、負の数のときだけ、数値データの後ろにマイナス記号を表示します。
^^^^	桁位置の指定の後ろに置いて、数値を指数形式で表示することを指定します。カレットが 4 つ (^^^^) なら D+xx、5 つ (^^^^^) なら D+xxx の形式で表示されます。符号の桁は左詰めになり、指数部はそれに合わせられます。先頭の +、末尾の + もしくは - を指定しない限り、スペースかマイナス符号を印字するために、1 桁が小数点の左にとられます。
¥¥	桁位置の指定の前に置いて、先頭に円マーク (¥) を表示することを指定します。¥ と数値 1 桁分の領域を示します。

**	小数点以上の桁をスペースで埋める代わりに、アスタリスク (*) で埋めます。円マークと同時に指定する場合は、 **¥ のようにアスタリスクを先に置きます。

出力する数値データは、指定された書式に合わせて丸められます。指定された書式では、対応する数値データが表示できないときは、パーセント記号(%) が表示され、そのことを示します。また、書式指定文字列で指定した桁数が 24 桁を超えると、"引数が有効範囲にありません" エラーになります。

その他の文字

その他の文字は、リテラル文字列として、そのまま出力されます。書式の指定に使われる文字をリテラル文字列として指定したい場合は、直前にアンダースコア (_) を挿入します。アンダースコアに続く文字は、すべてリテラル字列と解釈されます。アンダースコア自身を指定するには、アンダースコアを 2 つ続けてください。

参照

MSGBOX ステートメント，**PRINT** メソッド，**PRINT #** ステートメント，**LPRINT** ステートメント，**SPC** 関数，**TAB** 関数

PRINT # ステートメント

機能

数値または文字列データをシーケンシャル ファイルに書き込みます。指定した書式に従って出力することもできます。

構文

PRINT *#filenumber%*, [*expression*[{,|;}*expression*]...[{,|;}]]

PRINT *#filenumber%*, **USING** *formatstr$*; *expression*[; *expression*]...[;]

filenumber%	オープンしているシーケンシャル ファイルのファイル番号。
expression	出力するデータを示す数式または文字列式。省略すると、単に改行されます。複数の式を指定するときは、セミコロン (;) またはカンマ (,) で区切ります。
;	データを出力したあと、出力位置を進めないことを示します。次のデータは、この直後から出力されます。 **USING** キーワードを指定した場合、*expression* の間のセミコロンは単なる区切りで、最後のセミコロンだけが意味を持ちます。
,	データを出力したあと、次の出力領域 (14 桁ごと) に出力位置を進めることを示します。次のデータは、次の出力領域の先頭から出力されます。
formatstr$	出力するデータの書式を示す文字列式。

解説

フォームが表示されているときに SCRN: または CONS: に対して **PRINT #** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

PRINT # ステートメントは、出力先が画面ではなくシーケンシャル ファイルであることを除き、**PRINT** ステートメントと同じ機能です。また **USING** キーワードを使って書式を指定して出力するときは、**PRINT USING** ステートメントと同じ機能です。

PRINT # ステートメントでシーケンシャル ファイルにデータを書き込むときは、データが正しく出力されるように、データの区切りには十分注意する必要があります。データの区切りにカンマを使うと、表示したフィールド間のスペースが、そのままファイルに書き込まれます。

参照

PRINT ステートメント，**WRITE #** ステートメント

PRINTER オブジェクト

機能

PRINTER オブジェクトは特殊オブジェクトの 1 つで、**PRINTER** キーワードを指定することによってアクセスされます。テキストの出力を制御し、出力先を決めます。

構文

PRINTER.PrintTarget[= *setting%*]

PRINTER.*method*[*arguments*]

method	**PRINTER** オブジェクトに適用可能なメソッド。
setting%	PrintTarget プロパティの有効な設定値。
arguments	指定したメソッドに対する引数。

解説

メソッド

PRINT　　ENDDOC　　NEWPAGE

プロパティ

PrintTarget

PRINTER オブジェクトは、プリンタなどの出力先のデバイスを示し、出力を制御する特殊オブジェクトです。デフォルトでは PRN に対応しています。PrintTarget プロパティの設定によって、出力先を別のプリンタ ポートまたはファイルに変更できます。

PRINT メソッドで 1 ページを超える出力を印刷する場合は、バックグラウンドはそのページの最後で切り捨てられ、残りはテキストだけが印刷されます。

PRINTFORM メソッド

機能

フォームの ASCII ダンプをプリンタまたはファイルに出力します。

構文

[*formname.*]**PRINTFORM**

対象

フォーム

解説

PRINTFORM を使うと、そのとき表示されているコントロールや出力されたテキストを含むフォームのイメージを、プリンタまたはファイルに出力することができます。出力は、実際に表示されているイメージにできるだけ近いキャラクタに変換してから行われます。したがって、ファイルに出力した場合は、テキストファイルとして扱うことができます。

ウィンドウの影などは出力されません。

出力は、**PRINTER** オブジェクトの PrintTarget プロパティに設定されているプリンタまたはファイルに送られます。

デザイン時にフォームを印刷するには、フォーム デザイナの [ﾌｧｲﾙ] メニューから [印刷] コマンドを選択します。

参照

PrintTarget プロパティ

PrintTarget プロパティ

機能

PRINT メソッドおよび **PRINTFORM** メソッドの出力先のプリンタ デバイスまたはファイルを決めます。

構文

PRINTER.PrintTarget = *target$*

target$	出力先のプリンタ ポートまたはファイルを示す文字列式。デフォルトは LPT1 です。

対象

PRINTER オブジェクト

解説

設定できる値は、出力先のプリンタ ポート (LPT1, LPT2, LPT3) またはファイル名です。ファイル名を設定するとき、ドライブ名を含むフルパスでも、カレント ディレクトリからの相対指定でもかまいません。

参照

PRINT メソッド, **PRINTFORM** メソッド

PSET ステートメント

PRESET ステートメントを参照してください。

PUT ステートメント (ファイル I/O)

GET ステートメント (ファイル I/O) を参照してください。

PUT ステートメント (グラフィックス)

GET ステートメント（グラフィックス）を参照してください。

RANDOMIZE ステートメント, RND 関数

機能

RANDOMIZE ステートメントは、乱数ジェネレータを初期化 (乱数系列を再設定) する数値演算ステートメントです。**RND** 関数は、0～1 の範囲の単精度の乱数を返す数値演算関数です。

構文

RANDOMIZE [*seed*]

RND[(*n*)]

seed　乱数ジェネレータの初期化に使う、任意の式。省略すると、**TIMER** 関数の値が使われます。

n　**RND** 関数が次の乱数をどのように生成するかを決める値。通常は省略できます。n 引数に指定した値によって、次に発生する乱数は以下のようになります。

n < 0	指定した n に対して、つねに同じ値を返します。
n > 0 (または省略時)	乱数系列の次の乱数を返します。
n = 0	直前に生成した乱数を返します。

解説

RANDOMIZE ステートメントを実行せずに **RND** 関数を使うと、プログラムを実行するたびに同じ順番で乱数が返されます。異なる乱数系列を使いたい場合は、プログラム ループの先頭に **RANDOMIZE** ステートメントを挿入し、実行のたびに異なる引数を渡してください。

Microsoft BASIC バージョン 7.1 以前では、**RANDOMIZE** ステートメントの引数を省略すると、*seed* 引数に相当する値の入力を求めるプロンプトが表示されていました。Visual Basic では、*seed* 引数を省略すると、自動的に **TIMER** 関数の戻り値が使われます。

任意の範囲の整数の乱数を生成するには、次の式を使ってください。

```
INT ((upperbound - lowerbound + 1)*RND + lowerbound)
```

ここで、upperbound には範囲の上限の値を指定し、lowerbound は下限の値を指定します。

参照

TIMER 関数

READ ステートメント

DATA ステートメントを参照してください。

ReadOnly プロパティ

Archive プロパティを参照してください。

REDIM ステートメント

機能

動的配列に割り当てられているメモリ サイズの変更や、動的配列の宣言を行います。

構文

REDIM[**PRESERVE**][**SHARED**] *variable*(*subscripts*)[**AS** *type*]
[, *variable*(*subscripts*)[**AS** *type*]]...

PRESERVE	配列のサイズを変更しても、その配列にすでに格納されている値をそのまま残すことを示します。
SHARED	宣言された変数をモジュール内のすべてのプロシージャで共有することを示します。変数は、そのモジュールに対してグローバルになります。プロシージャの中で、個々に **SHARED** ステートメントを使う必要はありません。**SHARED** を指定できるのは、コード モジュールのモジュールレベル コード内だけです。
variable	宣言する変数の名前。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。
subscripts	各次元の添字の最大値と最小値。これにより、配列の次元数も決まります。次元数の最大値は 60 です。
type	変数のデータ型 (**INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, **STRING** * *n* またはユーザー定義型)。変数それぞれに対して指定します。省略すると、変数名に型宣言文字が付けられていればそのデータ型になり、付けられていなければ **SINGLE** (単精度浮動小数点数) と見なされます。

解説

subscripts 引数には、次の構文に従い、配列変数の次元と、各次元の添字の下限と上限を指定します。

[*lower* **TO**]*upper*[, [*lower* **TO**]*upper*]...

lower は添字の最小値を示し、*upper* は最大値を示します。*lower* のデフォルトは 0 ですが、**OPTION BASE** ステートメントを使うと、デフォルトを 1 に設定できます。

REDIM ステートメントは、動的配列として宣言された配列に割り当てられているメモリ領域の大きさを変更します。各次元の添字の最小値および最大値を変更することができます。

```
DIM A(50,50)                    ' $DYNAMIC
ERASE A
REDIM A(20,15)                  ' 配列 A は 2 次元のまま
```

ただし次元数を変更することはできません。次のステートメントは "配列の次元が無効です" エラーになります。

```
DIM A(50,50)                    ' $DYNAMIC
ERASE A
REDIM A(5,5,5)                  ' 次元数を 2 から 3 に変更
```

REDIM ステートメントは、すでに **DIM** ステートメントで添字を指定せずに (空のカッコだけを指定して) 宣言されていた動的配列の大きさを、設定または再設定するのにも使われます。この場合、**REDIM** ステートメントで指定できる次元数は最大 8 次元です。

また、**DIM** ステートメントの代わりに、配列変数を宣言することもできます。この場合、宣言された配列は必ず動的配列になります。

REDIM ステートメントは、ユーザー定義型の配列の宣言またはサイズの変更にも使えます。

REDIM ステートメントが実行されると、配列にすでにメモリが割り当てられていれば、そのメモリは解放され、新しいサイズのメモリが割り当てられます。**PRESERVE** キーワードを指定していなければ、数値配列のすべての要素は 0 に、また文字列配列のすべての要素は null 文字列に初期化されます。

PRESERVE キーワードを使うと、そのとき各要素に割り当てられている値を保持したまま、動的配列の大きさを変更できます。例えば、次のステートメントでは、配列 X() に格納されているデータを保持したまま、配列の 3 次元目に要素を増やすために領域が追加されます。

```
REDIM X(10, 10, 10)
.
.
.
REDIM PRESERVE X(10, 10, 15)
```

ただし **PRESERVE** キーワードを指定するとき、要素数や添字の上限，下限を変更できるのは最後の次元に対してだけです。プログラミング環境上で、最後以外の次元を変更しようとすると、"添字が有効範囲にありません" エラーが発生します。

REDIM PRESERVE ステートメントの後で **UBOUND** 関数を使うと、配列の中で値が格納されている要素数を求められます。

DIM ステートメントで **SHARED** キーワードが使われると、モジュール内のすべてのプロシージャの間で変数が共有されます。これに対して、**SHARED** ステートメントは、1 つのプロシージャとモジュールレベル コードの間で変数が共有されることを宣言します。

REDIM ステートメントは、モジュール レベルの **COMMON** ステートメントで **SHARED** キーワードを付けて宣言されてた動的配列の、次元数や各次元の添字の上限，下限を定義するために、Form_Load イベントに対するイベント プロシージャでよく使われます。

参照　**DIM** ステートメント，**COMMON** ステートメント，**OPTION EXPLICIT** ステートメント

REFRESH メソッド

機能

フォームやコントロールの再表示を行います。ファイル リスト ボックス, ディレクトリ リスト ボックス, ドライブ リスト ボックスの場合は、リストの内容が更新されます。

構文

[{[*formname*.]*ctlname*.|*formname*.}]**REFRESH**

対象

チェック ボックス, コンボ ボックス, コマンド ボタン, ディレクトリ リスト ボックス, ドライブ リスト ボックス, ファイル リスト ボックス, フォーム, フレーム, 水平スクロール バー, ラベル, リスト ボックス, オプション ボタン, ピクチャー ボックス, テキスト ボックス, 垂直スクロール バー

解説

一般に、フォームやコントロールは、イベントの処理がすべて終わった後に自動的に再表示されます。**REFRESH** メソッドは、これ以外のとき、例えばファイル リスト ボックス, ディレクトリ リスト ボックス, ドライブ リスト ボックスなどで、ファイルの状態が変化したときなどに便利です。

参照

AutoRedraw プロパティ

REM ステートメント

機能

プログラム中にコメントやメタコマンドを記述します。

構文

{**REM**|'} *text*

text コメントまたはメタコマンド。

解説

REM ステートメントは、ソースファイルとして保存されますが、メタコマンドとして解釈可能な場合を除いて、実行時やコンパイル時には無視されます。

REM キーワードの代わりに、シングル クォーテーション(') を使うことができます。シングル クォーテーション(') を使うと、他のステートメントの後ろに (コロンを使わずに) コメントを置くことができます。ただし **DATA** ステートメントの後ろにはコロンが必要です。

GOTO や **GOSUB** ステートメントの分岐先の行として、**REM** ステートメントの行を指定してもかまいません。この場合、**REM** ステートメントの後ろにある、最初の実行可能ステートメントに制御が移ります。

REMOVEITEM メソッド

機能　実行時にリスト ボックスまたはコンボ ボックスから項目を削除します。

構文　[*formname.*]*ctlname*.**REMOVEITEM** *index%*

ctlname	コンボ ボックスまたはリストボックスの名前。
index%	削除する項目のリスト中での位置を指定する整数。 0 が先頭を示します。

対象　リスト ボックス，コンボ ボックス

解説　リスト ボックス部分に表示されている項目に実行時にアクセスするには、List, ListCount, ListIndex の各プロパティを使ってください。

リスト ボックス部分に存在しない項目を *index%* 引数で指定すると、“引数が有効範囲にありません” エラーが発生します。

参照　**ADDITEM** メソッド

RESET ステートメント

機能　オープンされているすべてのファイルおよびデバイスをクローズします。

構文　**RESET**

参照　**CLOSE** ステートメント

Resize イベント

機能　フォームのサイズが変更されたとき、およびフォームが最初に表示されたときに発生します。

構文　**SUB Form_Resize ()**

対象　フォーム

解説　Resize イベントは、フォームのサイズが変更されたときに、その上に配置されているコントロールを移動したりサイズを変更したりするのに便利です。フォームのサイズに依存する変数やプロパティを再計算するために使ってください。

AutoRedraw プロパティが False (0) に設定されているフォームのサイズが変更されると、Resize イベントの後に Paint イベントが発生します。

フォームの Top, Left, Height, Width, ScaleHeight, ScaleWidth の各プロパティは、Resize イベントが発生したときに自動的に調整されます。

参照　AutoRedraw プロパティ，**LOAD** ステートメント，Paint イベント，**REFRESH** メソッド，**SHOW** メソッド

RESTORE ステートメント

DATA ステートメントを参照してください。

RESUME ステートメント

機能　エラー処理ルーチンを終了し、プログラムの実行を継続するエラー処理ステートメントです。

構文　**RESUME**[{*line*|**0**|**NEXT**}]

line　プログラムの実行を再開する行を示す、行ラベルか行番号。指定する行は、エラー処理ルーチンがプロシージャ中にあれば、同じプロシージャ中の行でなければなりません。また、エラー処理ルーチンがモジュールレベル コードにあれば、同じモジュール中の行でなければなりません。

0　エラー処理ルーチンを含むモジュール内の、最後に実行されたステートメントに戻ります。エラーが発生したモジュールの中でエラーが処理されたときは、プログラムの実行は、エラーが発生したステートメントから再開されます。引数を省略すると、 0 と見なされます。

NEXT　エラー処理ルーチンを含むモジュール内の、最後に実行されたステートメントの次のステートメントに戻ります。エラーが発生したモジュールの中でエラーが処理されたときは、プログラムの実行は、エラーが発生したステートメントの次のステートメントから再開されます。

解説

実行を再開する行は、エラーの発生した位置ではなく、エラーをトラップしたエラー処理ルーチンの位置に基づくことに注意してください。

エラー処理ルーチンを検索して、エラーが発生したプロシージャやモジュール以外に **RESUME** ステートメントを含むエラー処理ルーチンが見つかった場合は、そのプロシージャまたはモジュールで最後に実行されたステートメントは、そのプロシージャやモジュールから抜ける最後の呼び出しになります。

基本的に、そのモジュール レベルでのみエラーが発生するのでない限り、モジュールレベルのエラー処理ルーチンで **RESUME** *line* を使うのは避けてください。

エラー処理ルーチン以外で **RESUME** ステートメントが実行されると、"エラー未発生で RESUME は実行できません" エラーになります。

エラー処理ルーチンは、必ず **RESUME** ステートメントで終了しなければなりません。エラー処理ルーチンに制御が移り、**RESUME** ステートメントを実行する前に、**END, END SUB, END FUNCTION, END DEF** などが見つかると、"RESUME がありません" エラーになります。

エラー処理ルーチンを含むプログラムを BC コマンドラインからコンパイルするときは、必ず /E(On Error) または /X(Resume Next) オプションを指定してください。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

RGB 関数

機能

RGB それぞれの色を 0～255 で表した値を、Visual Basic for MS-DOS でフォームやコントロールの色の指定に使う 0～15 の値に変換します。

構文

RGB(*red%, green%, blue%*)

red%
green%
blue%　R, G, B それぞれの色の強度を表す 0～255 の整数。

RIGHT$ 関数

LEFT$ 関数を参照してください。

RMDIR ステートメント

CHDIR ステートメントを参照してください。

RND 関数

RANDOMIZE ステートメントを参照してください。

RSET ステートメント

LSET ステートメントを参照してください。

RTRIM$ 関数

LTRIM$ 関数を参照してください。

RUN ステートメント

機能

現在、メモリ上にあるプログラムをリスタートしたり、指定したプログラムを実行するフロー制御ステートメントです。

構文

RUN[*linenumber|filespec$*]

linenumber	プログラムの実行を始める行番号。省略すると、最初の実行可能な行と見なされます。
filespec$	.BAS ファイルまたは .EXE ファイルのパス名。拡張子を省略すると .BAS と見なされます。

解説

プログラミング環境で **RUN** ステートメントを実行する場合は、拡張子 .BAS のファイルのみ呼び出せます。この場合、拡張子は省略できます。ファイルが Basic のソースプログラムでなければ、エラーになります。拡張子 .EXE や .COM のファイルは呼び出せません。また、指定した .BAS のファイルが見つからない場合は、"ファイルが見つかりません" エラーが発生し、プログラミング環境に制御が戻ります。

プログラミング環境外で **RUN** ステートメントを実行する場合は、拡張子 .EXE のファイルに対してのみチェインでき、拡張子 .BAS のファイルにチェインすることはできません。

実行の開始位置を指定する場合、必ず行番号で指定しなければなりません。行ラベルと指定すると、エラーになります。エラーが発生した場合、プログラミング環境上で実行していればプログラミング環境へ戻り、コンパイルされた実行可能ファイルとして実行していればオペレーティング システムに制御が戻ります。

実行の開始行を指定するときは、その行はコード モジュールのモジュール レベルでなければなりません。フォーム モジュールのモジュール レベルには、実行可能なステートメントを置くことはできません。

SUB や FUNCTION プロシージャの中の **RUN** ステートメントでは、必ずモジュール レベルの行を指定しなければなりません。行を指定しないと、メイン モジュールの最初の実行可能な行から、実行が開始されます。

起動されたプログラムの実行が終了しても、制御は元のプログラムには戻りません。プログラムがプログラミング環境外で動作しているときは、制御はオペレーティング システムに戻ります。プログラミング環境で動作しているときは、プログラミング環境に戻ります。

RUN ステートメントを実行すると、すべてのファイルが閉じられ、プログラムに割り当てられたメモリは目的のプログラムのロードの前に消去されます。ファイルを開いたままで別のプログラムを実行したい場合は、**CHAIN** ステートメントを使用してください。

参照

CHAIN ステートメント

SADD 関数

機能

指定した文字列が格納されているメモリのオフセット アドレスを返すメモリ管理関数です。

構文

SADD(*stringvariable$*)

stringvariable$ 格納されている位置のオフセット アドレスを調べたい、単純文字列変数または文字列配列の要素。固定長文字列を指定することはできません。変数の値が null 文字列の場合は、戻り値は 0 になります。

解説

SADD 関数は、複数言語によるプログラムで、他言語で記述されたルーチンに far 文字列を渡す前に、アドレスを得る目的でよく使われます。

Basic は、プログラム実行中に文字列の格納位置を移動することがあるため、注意が必要です。

SADD 関数で得たアドレス (文字列の格納位置の先頭) の前や、その位置と **LEN** 関数の戻り値で調べたアドレス (文字列の格納位置の最後) の後ろに、Basic 以外の言語で書いたルーチンから、文字を追加しないでください。このような操作は、ランタイム エラーの原因となります。

文字列のセグメント アドレスを得るには **SSEG** 関数を使い、セグメントとオフセットの両方のアドレスを得るには **SSEGADD** 関数を使います。

従来の Basic では、**SADD** は near 文字列に対してしか使用できませんでした。Visual Basic は near 文字列をサポートしていません。

ScaleHeight, ScaleWidth プロパティ

機能

フォームまたはピクチャー ボックスの内側の高さ (ScaleHeight)、幅 (ScaleWidth) を返します。常に読み込み専用で、値を設定することはできません。

構文

[*formname.*][*ctlname.*]**ScaleHeight**

[*formname.*][*ctlname.*]**ScaleWidth**

ctlname ピクチャー ボックスの名前。

対象

フォーム，ピクチャー ボックス

解説

ScaleHeight と ScaleWidth プロパティの値は、常にキャラクタ単位で表されます。

ScaleHeight および ScaleWidth プロパティが返す値は、Height および Width プロパティが返す値から、境界やメニューの高さや幅を引いた値です。Height および Width プロパティや、関連するプロパティである Left および Top プロパティは、オブジェクトのサイズ変更や移動などの、オブジェクト自身の位置や領域の計算が必要なときに使います。これに対して ScaleHeight および ScaleWidth プロパティは、文字の出力や子コントロールの移動など、オブジェクト内部の領域の計算が必要なときに使います。

ScaleHeight および ScaleWidth プロパティが高さや幅を返す、オブジェクトの内部の領域を、“クライアント領域” と呼ぶことがあります。

参照　Height プロパティ，Width プロパティ

SCREEN オブジェクト

機能　**SCREEN** オブジェクトは特殊オブジェクトの 1 つで、**SCREEN** キーワードを指定することによってアクセスされます。画面の色やサイズの設定，全フォームの表示/非表示の切り換えなどに使います。

構文

SCREEN.*property*[= *setting%*]

SCREEN.*method*[*arguments*]

property	**SCREEN** オブジェクトが持つプロパティ。
setting%	プロパティの有効な設定値。
method	**SCREEN** プロパティに適用可能なメソッド。
arguments	指定したメソッドに対する引数

解説

メソッド

HIDE	SHOW

プロパティ

ActiveControl	ActiveForm	ControlPanel
Height	MousePointer	Width

SCREEN オブジェクトは、画面を示す特殊オブジェクトです。フォームを使わない場合は、**SCREEN** 関数や **SCREEN** ステートメントを使ってください。

SCREEN オブジェクトのプロパティは次のような情報の取得と設定を行います。

処理内容	プロパティ
ウィンドウ システムのデフォルト カラー	ControlPanel
画面のサイズ	Height, Width
フォーカスの位置の追跡	ActiveControl, ActiveForm

また、**SCREEN** オブジェクトに対して **HIDE** または **SHOW** メソッドを使うと、すべてのフォームの表示/非表示を 1 度に切り換えることができます。

参照　**SCREEN** 関数，**SCREEN** ステートメント

SCREEN ステートメント

機能　スクリーン モードを設定します。フォームが表示されている状態では利用できません。

構文　**SCREEN** *mode%*[, [*paletteswitch%*][, [*apage%*][, [*vpage%*][, *noinit%*]]]]

mode%	スクリーン モードを表す整数。
paletteswitch%	パレットのモードを表す整数。利用できるパレット番号と色値の指定できる範囲が変わります。デフォルトは 0 です。
apage%	テキスト表示やグラフィックス コマンドの対象となるグラフィックス スクリーン ページの番号。デフォルトは 0 です。
vpage%	表示するグラフィックス スクリーン ページの番号。デフォルトは 0 です。
noinit%	スクリーン モードを変更するとき、テキストおよびグラフィックス画面を初期化するかどうかを示す整数。 0 または 1 を指定でき、0 を指定すると自動的に初期化され、 1 を指定すると初期化しません。デフォルトは 0 です。

解説　フォームが表示されているときに **SCREEN** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。フォームを使っているアプリケーションでは、画面の各種の設定に **SCREEN** オブジェクトを利用してください。

SCREEN ステートメントは、スクリーン モード，パレット モード，スクリーン ページなど、画面に関する基本的な設定を行います。

指定できるスクリーン モードを示す番号，同時表示可能な色の数，スクリーン ページの枚数などは、使用しているコンピュータの機種に依存します。『機種依存情報』を参照してください。

通常、スクリーン モードを変更すると画面が初期化され、それまで表示されていた内容は消去されます。また、プログラムの起動時と終了時にも、自動的に画面の初期化が行われます。ただし *noinit%* 引数に 1 を指定すると、初期化せずにスクリーン モードを変更できます。また、最後に実行した **SCREEN** ステートメントで *noinit%* 引数に 1 が指定されていれば、画面の初期化を行わずにプログラムは終了します。プログラムの出力結果を表示したまま終了したいときや、画面の初期化をせずにパレットのモード (*paletteswitch%* 引数)，操作対象のグラフィックス スクリーン ページ (*apage%* 引数)，表示するグラフィックス スクリーン ページ (*vpage%* 引数) を変更したいとき、*noinit%* 引数に 1 を指定してください。

参照　**SCREEN** 関数，**SCREEN** オブジェクト

SCREEN 関数

機能

画面上の指定された位置にある文字の、文字コードまたは表示色を返します。フォームが表示されている状態では利用できません。

構文

SCREEN (*line%*, *column%*[, *colorflag%*])

line%	画面上の行位置。スクリーン モードで決まる画面行数以内の整数を指定してください。
column%	画面上の桁位置。1～80 の整数を指定してください。
colorflag%	知りたい情報を示す整数。省略するか 0 を指定すると文字コードを返し、1 を指定すると色番号を返します。グラフィックス モードでは無効です。

解説

フォームが表示されているときに **SCREEN** 関数を参照すると、"フォームが表示されているときは無効です" エラーが発生します。

colorflag% に 0 を指定するか省略したときに、*row%* と *column%* が指す位置が 2 バイト文字の 1 バイト目の場合は、その文字の漢字連続コードを返します。2 バイト文字の 2 バイト目を指しているときは、その文字の漢字連続コードに、マイナス符号を付けた値を返します。

例えば、画面の (10,10) にある文字が "A" のとき、次の **SCREEN** 関数を実行すると、文字 A の ASCII コードである 65 が返されます。

```
X = SCREEN(10,10)
```

また、画面の (1,1) にある文字が "あ" のとき、次の **SCREEN** 関数を実行すると、漢字連続コードを負にした値 -539 が返されます。

```
X=SCREEN(1,2)
```

colorflag% に 1 を指定すると、文字の表示色を得ることができます。返される値について詳しくは、『機種依存情報』を参照してください。

参照

COLOR ステートメント，**SCREEN** オブジェクト，**SCREEN** ステートメント

ScrollBars プロパティ

機能　テキスト ボックスが垂直および水平のスクロール バーを持っているかどうかを決めます。実行時には読み込み専用です。

構文　[*formname*.]*ctlname*.**ScrollBars**

ctlname　テキスト ボックスの名前。

対象　テキスト ボックス

解説　ScrollBars プロパティは、テキスト ボックスが垂直および水平のスクロール バーを持っているかどうかを示します。0 はなし，1 は水平スクロール バーのみ，2 は垂直スクロール バーのみ，3 は両方を示します。スクロール バーなし (0) がデフォルトです。

スクロール バーを表示するには、必ず MultiLine プロパティを True (-1) に設定しなければなりません。また、テキスト ボックスのサイズ (Width および Height プロパティ) がスクロール バーを表示できる最小サイズ以上でないと、スクロール バーは表示されません。

テキスト ボックスでは、方向キーや [HOME] キー，[END] キーなどは常に有効です。また、MultiLine プロパティが True (-1) に設定されていれば、[PAGE UP] キーや [PAGE DOWN] キーも有効です。これらのキーを使えば、スクロール バーがなくても、テキスト ボックスの内容をスクロールできます。

スクロール バーの色は、テキスト ボックスの BackColor プロパティの設定によって決まります。**SCREEN** オブジェクトの ControlPanel プロパティの SCROLLBAR_BACKCOLOR と SCROLLBAR_FORECOLOR 要素は影響しません。

参照　水平スクロール バー コントロール，垂直スクロール バー コントロール

SECOND 関数

機能　日付と時刻のシリアル値を評価して、0～59 までの範囲で、何秒かを返します。

構文　**SECOND**(*serial*)

serial　日付と時刻を表すシリアル値 (-53,688～65,380)。

SEEK 関数, SEEK ステートメント

機能

SEEK 関数は、データファイル中の、次に読み書きする位置を返します。**SEEK** ステートメントは、データファイル中の、次に読み書きする位置を設定します。

構文

SEEK(*filenumber&*)

SEEK [#]*filenumber%*, *position&*

filenumber&	オープンしているファイルのファイル番号。
filenumber%	オープンしているファイルのファイル番号。
position&	次に読み書きする位置を表す数式。 **RANDOM** モードでオープンしたファイルではレコード番号を指定します。 **BINARY, INPUT, OUTPUT, APPEND** の各モードでオープンしたファイルでは、ファイルの先頭からのバイト数を指定します。指定できる値の範囲は、1～2,147,483,647 (=2^{31}-1) です。

解説

SEEK 関数

SEEK 関数は、データ ファイル中で次に読み書きを行う位置を示す、1～2,147,483,647 (=2^{31}-1) の値を返します。**RANDOM** モードでオープンしたファイルでは、次に読み書きを行うレコードの番号を返します。**BINARY, OUTPUT, APPEND, INPUT** の各モードでオープンしたファイル (シーケンシャル ファイルまたはバイナリ ファイル) では、次に読み書きを行う位置を、ファイルの先頭バイトを 1 としたバイト数で返します。

SEEK 関数は、Visual Basic のデバイス (SCRN:, CONS:, KYBD:, COM*n*:, LPT*n*:, PIPE:) をサポートしていません。これらに対して実行すると、0 を返します。

次のプログラムは、最後の読み書きがファイルを 3 等分したときのどの部分に対して行われたかを示すメッセージを表示します:

```
SELECT CASE (SEEK(1))
    CASE IS < .333*LOF(1)
        PRINT "最初の 1/3"
    CASE .333*LOF(1) TO .667*LOF(1)
        PRINT "中央の 1/3"
    CASE IS >= .667*LOF(1)
        PRINT "最後の 1/3"
END SELECT
```

SEEK ステートメント

SEEK ステートメント実行後の入出力操作は、ここで設定した位置から行われます。ただし **GET** および **PUT** ステートメントにおけるレコード番号の指定は、**SEEK** ステートメントで設定した位置より優先します。

引数 *position&* に負の数または 0 を指定すると、"レコード番号が無効です" エラーが発生します。ファイルの終端より後ろの位置を指定して **SEEK** ステートメントを実行したあと、そのファイルに対して書き込みを行うと、ファイルは拡張されます。

SEEK ステートメントは、Visual Basic のデバイス (SCRN:, CONS:, KYBD:, COM*n*:, LPT*n*:, PIPE:) をサポートしていません。これらに対して実行しても無視されます。

参照

GET ステートメント (ファイル I/O), **OPEN** ステートメント (ファイル I/O), **PUT** ステートメント (ファイル I/O)

SELECT CASE ステートメント

機能

条件式の値にしたがって、複数のステートメント ブロックのいずれかを実行させるフロー制御ステートメントです。

構文

SELECT CASE *testexpression*

CASE *expressionlist-1*

[*statementblock-1*]

[**CASE** *expressionlist-2*

[*statementblock-2*]]

[**CASE** *expressionlist-n*

[*statementblock-n*]...]

[**CASE ELSE**

[*statementblock-n*]]

END SELECT

testexpression	どのステートメント ブロックを実行するかを決める数式または文字列式。
expressionlist-1	*testexpression* と同じデータ型の、1 つ以上の条件。
...	
expressionlist-n	
statementblock	任意の行数のステートメント。

解説

SELECT CASE ステートメントは、いくつかのステートメントブロックの中から、指定した条件に一致する 1 つだけを実行します。*testexpression* が **CASE** 節のいずれかの *expressionlist* と一致すると、次の **CASE** 節または **END SELECT** ステートメントまでのステートメントブロックが実行されます。ブロックの実行が終わると、**END SELECT** の次のステートメントから実行が続けられます.

expressionlist には、次の 3 つの書式のうちのいずれかに従って、条件を指定します。

expression[, *expression*...]	一致による条件
expression **TO** *expression*	範囲による条件
IS *relational-operator expression*	関係による条件

expression は数式または文字列式です。データ型は、*testexpression* 引数と同じか、*testexpression* の型を強制的に変換できる型でなければなりません。例えば、*testexpression* が整数型のときに、*expression* に倍精度浮動小数点数を指定することはできますが、文字列型を指定することはできません。

relational-operator は、 <, <=, >, >=, <>, = のいずれかの演算子です。

TO キーワードを使って範囲を指定するときは、小さい方を先に指定してください。例えば、CASE -1 TO -5 と指定したとき、*testexpression* が -4 であっても一致したと見なされません。必ず、CASE -5 TO -1 と指定してください。

IS キーワードを指定すると、関係演算子を使うことができます。

CASE ELSE 節を指定すると、*testexpression* がどの **CASE** 節にも一致しなかったときに、**END SELECT** までのステートメント ブロックが実行されます。*testexpression* の値が予測できないときは、**CASE ELSE** 節を使ってください。

CASE ELSE ステートメントがなく、*testexpression* がどの **CASE** 節にも一致しないとき、**END SELECT** の次のステートメントから実行が続けられます。

次に示すように、それぞれの **CASE** 節で、複数の式や範囲を指定することができます。

```
CASE 1 TO 4, 7 TO 9, 11, 13, IS > MaxNumber%
```

また次のように、文字列の範囲を示す式を指定することもできます。この **CASE** 節では、"everything", TestItem$ の現在の値, アルファベット順で nuts と soup の間にある文字列のいずれかの文字列が一致します。

```
CASE "everything", TestItem$, "nuts" TO "soup"
```

文字列は、それぞれの文字の文字コード (ASCII コードまたは漢字連続コード) によって比較されます。文字コードでは、小文字は大文字よりも大きい値を持つため、次のような関係となります。

```
nuts > Nuts > NUTS
```

CASE 節に指定された条件は、最初の **CASE** 節から順にテストされます。*textexpression* に指定した式の値が、複数の **CASE** 節の条件に一致するときでも、最初に一致した **CASE** 節のステートメントブロックだけが実行されます。

SELECT CASE ステートメントはネストできます。つまり **SELECT CASE** ブロックの中に、別の **SELECT CASE** ブロックを記述することができます。このとき、各 **SELECT CASE** ステートメントには、それぞれ対応する **END SELECT** ステートメントがなければなりません。

参照 **IF...END IF** ステートメント, **IF TYPEOF** ステートメント

SelLength, SelStart, SelText プロパティ

機能

テキスト ボックスまたはコンボ ボックスのテキスト ボックス部分で、選択 (反転表示) されているテキストの文字数 (SelLength)，開始位置 (SelStart)，文字列 (SelText) を取得または設定します。デザイン時には無効です。

構文

[*formname.*]*cltname*.**SelLength**[= *length%*]

[*formname.*]*cltname*.**SelStart**[= *point%*]

[*formname.*]*cltname*.**SelText**[= *string-expression$*]

ctlname	テキスト ボックスまたはコンボ ボックスの名前。
length%	選択するテキストの文字数を示す整数。
point%	選択するテキストの開始位置を先頭からの文字数で示した整数。
string-expression$	挿入または置き換えるテキストを示す文字列式。

対象

コンボ ボックス，テキスト ボックス

解説

SelLength, SelStart, SelText の各プロパティは、テキスト ボックスまたはコンボ ボックスのテキスト ボックス部分で、次のような操作を行うために使います。

- カーソル位置を取得または設定する。
- そのとき選択されている文字列を取得する。
- そのとき選択されている文字列を別の文字列へ置き換える。
- 任意の位置に文字列を挿入する。
- 文字列を消去する。

テキストが選択されていないとき、SelStart プロパティはそのときのカーソル位置を示します。また、SelText プロパティの値は null 文字列 ("") です。

SelLength および SelStart プロパティに負の値を設定しようとすると、エラーが発生します。

SelStart プロパティに、そのとき表示されている文字列の文字数より大きい値を設定すると、テキストの文字数と見なされます。このとき、表示されているテキストの末尾がカーソル位置になります。

SelStart プロパティの値を変更すると、SelLength プロパティの値は 0 にリセットされます。

コード中から SelText プロパティに文字列を設定すると、カーソル位置 (SelStart プロパティの値) にその文字列が挿入されます。そのとき文字列が選択されていたら、新しい文字列に置き換えられます。SelStart, SelLength, SelText の順で設定すると、任意の位置の文字列を置き換えることができます。挿入や置き換えが起こると、カーソルは挿入

または置き換えたテキストの先頭に移動し、SelLength プロパティの値は 0 にリセットされます。

参照　Text プロパティ

Separator プロパティ

機能　メニュー項目をセパレータ バーとして表示するかどうかを決めます。

構文　[*formname*.]*ctlname*.**Separator**[= *boolean%*]

ctlname	メニュー コントロールの名前。
boolean%	セパレータ バーとして表示されるかどうかを示す値。 True (-1) は表示されることを示し、False (0) は表示されないことを示します。デフォルトは False (0) です。

対象　メニュー

解説　デザイン時には、メニュー デザイン ウィンドウで“セパレータ”チェック ボックスをチェックするか、“キャプション”欄に "-" と入力すると、Separator プロパティは自動的に True (-1) に設定されます。Separator プロパティが True (-1) に設定されているメニュー コントロールは、イベントを認識できません。この状態のメニュー コントロールに対し、実行時に Separator プロパティを False (0) に変更すると、イベントが認識できる状態になります。このとき、コマンド名としては "-" が表示されます。

デザイン時に、通常のメニュー項目として作成されたメニュー コントロールの Separator プロパティは、False (0) になっています。この状態のメニュー コントロールに対し、実行時に Separator プロパティを True (-1) に変更すると、イベントを認識できなくなります。このとき、コマンド名としては "-" が表示されます。設定を False (0) に戻すと、再びデザイン時に設定したコマンド名が表示され、イベントが認識できるようになります。

トップ レベルのメニュー項目の Separator プロパティを True (-1) に設定しようとすると、エラーが発生します。

参照　Checked プロパティ，Enabled プロパティ，Visible プロパティ

SetFEPMode ルーチン

機能　日本語入力システムの状態を設定するルーチンです。

構文　**CALL SetFEPMode** (*FEPSW%*)

FEPSW%　設定したい状態を示す整数。0 を指定すると日本語入力システムはオフになり、0 以外の値を指定するとオンになります。

解説　このルーチンをプログラミング環境上で利用するには、何らかの Quick ライブラリをロードしている必要があります。VBDOS 起動時に /L オプションを指定してください。また、**DECLARE** ステートメントで SUB プロシージャとして宣言されている必要があります。次のように宣言します。

```
DECLARE SUB SetFEPMode (BYVAL FEPSW%)
```

この宣言は、インクルード ファイル VBDOS.BI にも含まれています。

フォームが表示されている場合、日本語入力システムをオンにできるのは、フォーカスがテキスト ボックスか、Style プロパティが 0 または 1 のコンボ ボックスにあるときだけです。それ以外のコントロールまたはフォームにフォーカスがあるときに SetFEPMode ルーチンを呼び出すと、次にフォーカスがテキスト ボックスまたはコンボ ボックスに移ったときに、日本語入力システムがオンまたはオフになります。

日本語入力システムの状態を取得するには、GetFEPMode% ルーチンを使います。

日本語入力システムがインストールされていないときにも SetFEPMode ルーチンを呼び出すことは可能ですが、呼び出しても単に無視されます。

日本語入力システムは、MS-KANJI API に準拠していなければなりません。

参照　GetFEPMode% ルーチン，**SETFOCUS** メソッド，テキスト ボックス，コンボ ボックス，**$INCLUDE** メタコマンド

SETFOCUS メソッド

機能　指定したフォームまたはコントロールにフォーカスを移します。

構文　[{[*formname.*]*ctlname.*|*formname.*}]**SETFOCUS**

対象　チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，水平スクロール バー，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

SETFOCUS メソッドは、指定したフォームやコントロールがユーザーからの入力を受け取るように、そのフォームまたはコントロールにフォーカスを移します。

コントロールが配置されているフォームに対して **SETFOCUS** メソッドを使うと、そのフォームのデフォルト コントロールにフォーカスが移ります。

子フォームを持つフォームに対して **SETFOCUS** メソッドを使うと、子フォームまたは子フォーム上のコントロールにフォーカスが移ります。

Enabled プロパティが False (0) に設定されているオブジェクトや非表示になっているオブジェクトは、フォーカスを持てません。

フォーカスを持つことができないコントロールに対して **SETFOCUS** メソッドを使おうとすると、"引数が有効範囲にありません" エラーが発生します。

オブジェクトを指定せずに **SETFOCUS** メソッドを使うと、その **SETFOCUS** メソッドを含むフォーム モジュールのフォームにフォーカスが移ります。

参照

GotFocus イベント，LostFocus イベント

SETMEM 関数

機能

far ヒープ (far オブジェクトが格納されている領域) で使うメモリ容量を変更し、それに関する情報を返すメモリ管理関数です。

構文

SETMEM(*numeric-expression&*)

numeric-expression& far ヒープを拡大または縮小するバイト単位の量。拡大するときは正の値、縮小するときは負の値を指定します。 0 を指定すると far ヒープのサイズは変更されず、そのときのサイズが戻り値になります。

解説

SETMEM 関数は、変更後の far ヒープ全体のバイト数を返します。現在の far ヒープのサイズを得たいときは、引数に 0 を指定します。

Visual Basic は、プログラムを起動するとき、できるだけ多くのメモリを far ヒープに割り当てます。このため最初に **SETMEM** 関数を呼び出したときには、far ヒープを拡大できません。

引数で指定した大きさまで拡大できないときは、できるだけ大きなサイズのメモリを割り当てます。

複数言語によるプログラムでは、他の言語で書かれたルーチンが far メモリを動的に利用できるようにするため、**SETMEM** 関数を使って far ヒープ領域を縮小することができます。

SETTEXT メソッド

機能　**CLIPBOARD** オブジェクトにテキスト文字列を格納します。

構文　**CLIPBOARD.SETTEXT** *string-expression$*

string-expression$　CLIPBOARD に格納される文字列データ。

対象　**CLIPBOARD** オブジェクト

解説　**CLIPBOARD** オブジェクトとそれに適用可能なメソッドを使うと、コード中から **CLIPBOARD** オブジェクトの内容を操作できます。テキスト ボックスやコンボ ボックスの SelStart, SelLength, SelText プロパティと組み合わせて使うことによって、メニューやコマンド ボタンでカット & ペーストの機能を実現できます。

参照　**CLEAR** メソッド，**GETTEXT** メソッド

SetUEvent ルーチン

機能　ユーザー定義のイベントを引き起こす Basic のエントリ ポイントを設定します。

構文　**SetUEvent**

解説　このルーチンで扱うイベントは、フォームやコントロールを使ったイベント ドリブン プログラミング モデルのイベントとは異なることに注意してください。フォームやコントロールに関連したイベントには影響を与えません。ただし、フォームやコントロールを使ったアプリケーションでも使うことができます。

SetUEvent は、**ON UEVENT** で設定されたイベント処理ルーチンに対して、イベントが発生したことを知らせます。

SetUEvent ルーチンは Basic の一部で、プログラムをコンパイルするか、/L オプションを指定して VBDOS.EXE を起動すると、自動的 Basic の中に組み込まれます。割り込みサービス ルーチンは、SetUEvent ルーチンを呼び出さなければなりません。これは、イベントが起こったことをプログラムに知らせる唯一の方法です。SetUEvent ルーチンは、アセンブリ言語だけではなく、どんな言語プログラムからでも呼び出せます。

参照　**ON UEVENT** ステートメント，**UEVENT** ステートメント

SGN 関数

機能　指定した数式の符号を示す値を返す、数値演算関数です。

構文　**SGN**(*numeric-expression*)

numeric-expression　任意の数式。

解説　引数が正なら 1、負なら -1、0 なら 0 を返します。

参照　**ABS** 関数

SHARED ステートメント

機能　モジュール レベルで宣言した変数を、引数として引き渡さずに、SUB または FUNCTION プロシージャ内で利用できるようにします。

構文　**SHARED** *variable*[()][**AS** *type*][, *variable*[()][**AS** *type*]]...

variable　モジュール レベルで宣言されている単純変数または配列変数。配列を指定するときは、空のカッコを後ろに付けます。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。

type　変数のデータ型 (**INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, **STRING** * *n* またはユーザー定義型)。変数それぞれに対して指定します。省略すると、変数名に型宣言文字が付けられていればそのデータ型になり、付けられていなければ **SINGLE** (単精度浮動小数点数) と見なされます。

解説　**SHARED** ステートメントは、1 つのプロシージャとモジュールレベル コードの間で変数を共有することを宣言します。これに対して、**COMMON** または **DIM** ステートメントで **SHARED** キーワードを指定して宣言すると、変数はそのモジュール内のすべてのプロシージャで共有されます。

SHARED ステートメントが使えるのは、SUB または FUNCTION プロシージャの中だけです。

SHARED ステートメントで変数が共有できるのは、同じモジュール内だけです。Quick ライブラリにあるプログラムや、別々にコンパイルしてリンクしたプロシージャとの間では、変数を共有できません。**SHARED** ステートメントは、同じモジュール内にあるモジュールレベルコードと SUB, FUNCTION プロシージャの間で変数を共有します。

参照　**COMMON** ステートメント，**DIM** ステートメント

SHELL ステートメント

機能

実行中の Visual Basic プログラムを中断し、.COM, .EXE, .BAT の各プログラムや、MS-DOS のコマンドを実行します。実行が終了すると、**SHELL** ステートメントの直後のステートメントから、Visual Basic プログラムの実行を再開します。

構文

SHELL[*commandstring$*]

commandstring$　実行可能ファイルの名前や MS-DOS のコマンド名と、そのパラメータ。

解説

commandstring$ 引数を省略すると、新しいシェル (COMMAND.COM) が起動され、MS-DOS のプロンプトが表示されて、コマンドが入力できる状態になります。Visual Basic に戻るには、EXIT コマンドを入力します。

SHELL ステートメントによって起動される .COM, .EXE, .BAT ファイルおよび MS-DOS コマンドは、“子プロセス”と呼ばれます。**SHELL** ステートメントは、COMMAND.COM を /C オプション付きで呼び出し、その上で子プロセスを実行します。このとき /C オプションによって、*commandstring$* で指定したパラメータが子プロセスに渡されます。このオプションは、標準入出力をリダイレクトしたり、DIR や PATH などの内部コマンドを実行するときにも使います。

commandstring$* 中のプログラム名には、任意の拡張子を付けることができます。拡張子を省略すると、.COMファイル，.EXE ファイル，.BAT ファイルの順番でファイルを検索します。COMMAND.COM が見つからないときは、"ファイルが見つかりません" エラーになります。*commmandstring$ で指定したファイルが見つからない場合、Visual Basic ではエラーになりませんが、MS-DOS がエラーを発生します。

実行可能ファイルの名前の後ろには、スペースを置いたあとで、子プロセスに渡すパラメータを指定します。

子プロセスを実行している間も、Visual Basic はメモリ上に存在しています。したがって、メモリが不足することがありますので注意してください。子プロセスが終了すると、Visual Basic は処理を継続します。

SHOW メソッド

機能

フォームを (まだロードされていなければ) ロードして表示し、そのフォームをアクティブにします。

構文

[*formname.*]**SHOW** [*style%*]

SCREEN.SHOW

style%　フォームがモーダルかモードレスかを指定する定数。 0 がモードレスを示し、1 がモーダルを示します。省略すると 0 と見なされます。**SCREEN** オブジェクトに対しては指定できません。

対象　フォーム，**SCREEN** オブジェクト

解説　**SCREEN.SHOW** を使うと、**SCREEN.HIDE** によって非表示になっていたすべてのフォームを表示できます。

SHOW メソッドで指定したフォームがロードされていなければ、そのフォームは自動的にロードされてから表示されます。

Load イベントに対するイベント プロシージャ中で、モーダルなフォームとして表示するために、*style%* 引数に 1 を指定して **SHOW** メソッドを使うときには注意が必要です。モーダルなフォームは、そのフォームを明示的に非表示にするかアンロードするまで、あるいはコード中から別のモーダルなフォームを表示するまで、他のフォームはユーザーからの入力を受け付けず、それをイベントとして認識しません。また、イベント プロシージャの **SHOW** メソッドの行以降に記述したコードも実行されません。したがって、フォームをモーダルとして表示する場合は、必ずそのフォームを明示的に非表示あるいはアンロードする方法を用意しておかなければなりません。通常は、そのフォーム上にコマンド ボタンを置き、ユーザーがそのボタンをクリックしたときに、非表示またはアンロードするようにします。

モーダルなフォームのフォーム コードから、別のフォームをモーダルとして表示することができます。この場合、新しく表示されたフォームが非表示またはアンロードされると、元のモーダルなフォームがアクティブになります。

style% 引数に 1 を指定しなければ、**SHOW** メソッドの行以降の書かれているコードは続けて実行されます。

MSGBOX または **INPUTBOX$** 関数で表示されるダイアログ ボックスを、ロード，アンロードしたり、**SHOW** メソッドを使って表示することはできません。

参照　**HIDE** メソッド，**LOAD** ステートメント，**UNLOAD** ステートメント

SIN 関数

COS 関数を参照してください。

SLEEP ステートメント

機能

プログラムの実行を一時中断するフロー制御ステートメントです。フォームが表示されている状態では利用できません。

構文

SLEEP [*seconds%*]

seconds% 中断する時間を秒単位で示す整数。

解説

フォームが表示されているときに **SLEEP** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

イベント (**ON COM** や **ON TIMER** など) が発生しても、対応するトラップが有効になっていなければ、**SLEEP** による停止に割り込むことはできません。**ON** *event* ステートメントでトラップを初期化し、*event* **ON** ステートメントで有効にし、さらに、*event* **OFF** や *event* **STOP** ステートメントでイベントの発生が禁止されていないことが必要です。

seconds& を省略したり、0 や整数でない値を指定した場合は、何かのキーが押されるか、トラップが有効になっているイベントが発生しない限り、プログラムは停止したままになります。

SLEEP は、実行が開始された後のキー入力にのみ反応します。実行前に入力されてバッファに保持されていたキー入力は無視されます。

参照

WAIT ステートメント

SmallChange プロパティ

LargeChange を参照してください。

Sorted プロパティ

機能

コンボ ボックスやリスト ボックス中の項目が自動的にソートされるかどうかを決めます。実行時には読み込み専用です。

構文

[*formname.*]*ctlname*.**Sorted**

ctlname コンボ ボックスまたはリスト ボックスの名前。

対象

コンボ ボックス，リスト ボックス

解説

Sorted プロパティは、コンボ ボックスやリスト ボックスに表示される項目が、自動的にソートされるかどうかを示します。True (-1) に設定すると、ソートされます。このとき、アルファベットは大文字小文字を区別せずにアルファベット順に並べられ、2 バイト文字は文字コード順に並べられます。False (0) に設定すると、ソートされずに表示されます。デフォルトは False (0) です。

Sorted プロパティを True (-1) に設定すると、表示内容が変更されたときにも、自動的にアルファベット順に並べ替えられます。また、リスト ボックス部分の各項目に順に割り振られているインデックス番号 (配列の添字) も、表示順序に合わせて自動的に変更されます。

Sorted プロパティが True (-1) に設定されている場合は、**ADDITEM** メソッドで項目を追加するときに *index%* 引数を指定しないように注意してください。*index%* を省略すると、新しい項目はソートされた正しい位置に挿入されます。もし *index%* で指定した位置がソートされた正しい位置と一致しないと、それ以降に追加される要素は正しくソートされなくなります。

参照

ADDITEM メソッド，List プロパティ，ListCount プロパティ，ListIndex プロパティ，**REMOVEITEM** メソッド

SOUND ステートメント

機能

スピーカーから音を発生させます。

構文

SOUND *frequency, duration*

frequency	音の周波数 (Hz) を表す、38～32767 の整数。
duration	音を継続させる時間を表す、0～65535 の整数または単精度浮動小数点数。約 0.055 秒を 1 とする内部単位を使います。1 秒が、18.2 にあたります。

解説

duration に 0 を指定すると、現在バックグラウンドで実行されている **SOUND** ステートメントが中断されます。**SOUND** ステートメントを実行していないときは何も起こりません。

参照

BEEP ステートメント，**PLAY** ステートメント (ミュージック)

SPACE$ 関数

機能　指定した個数のスペースからなる文字列を返す、文字列処理関数です。

構文　**SPACE$**(*n*%)

n%　スペースの数を表す、0～32,767 の整数。

SPC 関数

機能　**PRINT, LPRINT, PRINT #** ステートメントおよび **PRINT** メソッド中で使い、指定した個数のスペースを返します。

構文　**SPC**(*n*%)

n%　挿入したいスペースの数を表す 0～32767 の整数。

解説　**SPC** 関数は、*n*% で指定した個数のスペースに置き換えられます。したがって画面上では、前に表示されていた文字は上書きされます。結果としてテキスト カーソルが、現在の出力位置から指定した桁数分だけ右に移動します。

n% の値が 1 未満のとき、*n*% は無効です。

SPC 関数の動作は次の 3 つの値に依存します。

- 引数 *n*%
- **SPC** 関数を実行したときの出力行上の出力位置
- そのときの出力行の桁数

出力デバイスの桁数は **WIDTH** ステートメントを使って設定できます。

n% にそのときの出力行の桁数以上の値を指定すると、桁数で割った余りと見なされます。つまり次のように計算されます。

SPC(*n*% **MOD** 桁数)

n% にそのときの出力行の桁数未満の値を指定し、現在の出力位置が行の途中にある場合、行の右端まで達したら、次の行に折り返します。

スペースは 1 バイト文字 (ASCII コード &H20) です。

参照　**LPRINT** ステートメント，**PRINT** メソッド，**PRINT** ステートメント，**PRINT #** ステートメント，**SPACE$** 関数，**TAB** 関数

SQR 関数

機能　引数に指定した値の平方根を返す数値演算関数です。

構文　**SQR**(*numeric-expression*)

numeric-expression　平方根を求める、0 以上の値。

SSEG 関数

機能　指定した文字列変数が格納されているセグメント アドレスを返す、メモリ管理関数です。

構文　**SSEG**(*stringvariable$*)

stringvariable$　格納されている位置のセグメント アドレスを調べたい、単純文字列変数または文字列配列の要素。固定長文字列を指定することはできません。変数の値が null 文字列の場合は、戻り値は 0 になります。

解説　**SSEG** 関数は、複数言語によるプログラムで、他言語で記述されたルーチンに far 文字列を渡す前に、その文字列の far アドレスを得る目的でよく使われます。

文字列のオフセット アドレスを得るには **SADD** 関数を使い、セグメントとオフセットの両方のアドレスを得るには **SSEGADD** 関数を使います。

固定長文字列のセグメント アドレスは、**VARSEG** 関数で調べることができます。

参照　**DEF SEG** ステートメント，**SSEGADD** 関数

SSEGADD 関数

機能

指定した文字列変数が格納されているセグメントおよびオフセット アドレスを返す、メモリ管理関数です。

構文

SSEGADD(*stringvariable$*)

stringvariable$ 格納されている位置のアドレスを調べたい、単純文字列変数または文字列配列の要素。固定長文字列を指定することはできません。変数の値が null 文字列の場合は、戻り値は 0 になります。

解説

SSEGADD 関数は、複数言語によるプログラムで、他言語で記述されたルーチンに far 文字列を渡す前に、その文字列の far アドレスを得る目的でよく使われます。

SSEGADD 関数は、文字列のセグメントとオフセットの両方のアドレスを 1 個の長整数 (4 バイト) として返します。

文字列のオフセット アドレスを得るには **SADD** 関数を使い、セグメント アドレスを得るには **SSEG** 関数を使います。

固定長文字列のセグメント アドレスは、**VARSEG** 関数で調べることができます。

参照

DEF SEG ステートメント，**SSEG** 関数

STACK ステートメント

機能

スタックの大きさを再設定します。

構文

STACK [*longinteger&*]

longinteger& スタックに対して割り当てるバイト数。省略すると、スタックはデフォルトの大きさ (約 3K バイト) に設定されます。0 を指定すると、最小のスタック サイズ (約 0.325K バイト) に設定されます。

解説

要求するスタック領域が大きすぎると、**STACK** ステートメントは確保できる最大限のスタック領域を確保しようとします。

STACK ステートメントは、コード モジュール (.BAS) のモジュール レベルでのみ使用可能です。

STACK ステートメントは、SUB プロシージャや FUNCTION プロシージャが再帰呼び出しや何レベルにもわたって呼び出しを行うプログラムで効果を発揮します。

参照

STACK 関数

STACK 関数

機能　割り当て可能な最大のスタック サイズ (DGROUP の未使用領域) を返す関数です。

構文　**STACK**

解説　スタック領域の残り容量を知るには、FRE(-2) を使用します。

参照　**STACK** ステートメント

$STATIC メタコマンド

$DYNAMIC メタコマンドを参照してください。

STATIC ステートメント

機能　単純変数や単純配列を、SUB, FUNCTION プロシージャまたは DEF FN ブロックに対してローカルな変数として宣言し、呼び出しの間変数の値を保護するステートメントです。

構文　**STATIC** *variable*[()][**AS** *type*][, *variable*[()][**AS** *type*]]...

variable　プロシージャや DEF FN ブロックに対してローカルにしたい変数。配列を指定するときは、空のカッコを後ろに付けます。

type　変数のデータ型 (**INTEGER**, **LONG**, **SINGLE**, **DOUBLE**, **CURRENCY**, **STRING**, **STRING** * *n* またはユーザー定義型)。変数それぞれに対して指定します。省略すると **SINGLE** (単精度浮動小数点数) と見なされます。

解説　**STATIC** ステートメントは、単純変数または配列変数を、プロシージャまたは DEF FN ブロックに対してローカルであると宣言します。これにより、プロシージャ呼び出しの間、変数の値を保護します。ローカル変数は、次にプロシージャが呼び出されるまで、変数の値を保存します。

STATIC ステートメントは、SUB または FUNCTION プロシージャ，DEF FN ブロックの中だけで宣言できます。**STATIC** ステートメントで宣言した変数は、モジュールレベルの **DIM**, **REDIM** または **COMMON** ステートメントで **SHARED** キーワードを付けて宣言された同じ名前の変数より優先します。同様に、同じ名前のグローバル定数よりも優先します。つまり同じ名前であっても別の変数として扱われます。

通常、DEF FN ブロックで使う変数は、そのモジュールのモジュールレベル コードと共有されます。DEF FN ブロックで **STATIC** ステートメントを使うと、その変数を DEF FN ブロック内だけで有効なローカル変数として宣言できます。

STATIC ステートメント，**STATIC** キーワード，**$STATIC** メタコマンドは、いずれも変数や配列に影響を与える目的で使われますが、それぞれ機能は異なります。

種類	機能
SUB, FUNCTION ステートメント中の **STATIC** キーワード	プロシージャ内の変数がデフォルトで静的であることを宣言します。モジュール レベルで共有が宣言されている名前を持つ変数は、モジュール レベルと共有されます。
STATIC ステートメント	指定した変数が静的であることを宣言します。モジュールレベルで共有が宣言されている名前より優先します。
$STATIC メタコマンド	配列がデフォルトで静的であることを宣言します。

参照

DEF FN ステートメント，**FUNCTION** ステートメント，**$STATIC** メタコマンド，**SUB** ステートメント

STOP ステートメント

機能

プログラムの実行を中止するフロー制御ステートメントです。

構文

STOP[*n*%]

n% プログラムを中断したときに、オペレーティング システムに返す値。省略すると、0 と見なされます。

解説

STOP ステートメントは、プログラムの実行を停止させます。プログラム中のどこにでも置くことができます。

n% 引数が有効なのは、コンパイルした実行可能ファイルをプログラミング環境外で実行し、正常終了したときだけです。トラップされなかったエラーや致命的なエラーでプログラムの実行が中止されたときは、*n*% 引数の指定にかかわらず -1 が返されます。またプログラミング環境では、つねにエラーレベルは 0 です。

/O オプションを付けてコンパイルした独立型の実行可能 (.EXE) ファイルでは、**STOP** ステートメントはすべてのファイルをクローズし、オペレーティング システムへ戻ります。このとき *n*% を返します。プログラミング環境上では、ファイルをオープンしたままプログラミング環境に戻ります。

BC コンパイラの /D, /E, /X などのオプションを指定してコンパイルしたときは、プログラムに行番号が付いてる場合、**STOP** ステートメントは、プログラムの実行を停止した行の行番号を表示します。**STOP** ステートメントのある行に行番号がなければ、直前の行番号を表示します。プログラムに行番号が付いていないときは、0 を表示します。

以前のバージョンの Basic では、デバッグを行うために **STOP** ステートメントを使っていましたが、Visual Basic プログラミング環境のデバッグ機能を使えば、この目的で **STOP** ステートメントを使う必要はありません。

参照　**END** ステートメント，**SYSTEM** ステートメント

STR$, VAL 関数

機能　**STR$** 関数は、数式の値を文字列で表現した値を返す文字列処理関数です。**VAL** 関数は、文字列で表現された値を数値に変換する文字列処理関数です。

構文

STR$(*numeric-expression*)

VAL(*string-expression$*)

numeric-expression	文字列に変換する値を示す数式。
string-expression$	数値に変換する文字列を示す文字列式。

解説　**STR$** 関数に正の引数を指定すると、返される文字列の先頭には 1 文字の空白が含まれます。

VAL 関数の引数には、数値を表現している文字列を指定してください。数値として認識できない文字が現れると、その位置で読み込みを中止します。また、引数に含まれる空白文字，タブ，ラインフィードは無視されます。例えば、次の関数の値は 1615198 になります。

```
VAL("    1615 198th Street")
```

STRING$ 関数

機能　指定した文字コードの示す文字か、文字列の先頭文字を、指定した文字数だけ並べた文字列を返す文字列処理関数です。

構文

**STRING$**(*length%*, {*char-code%* | *string-expression$*})

length%	返される文字列の文字数を表す数式。
char-code%	ASCII コードまたは漢字連続コードを表す 0～11,535 の整数。
string-expression$	任意の文字列式。先頭の 1 文字だけが意味を持ちます。

StringAddress ルーチン

機能

Visual Basic の可変長文字列データを指す far ポインタを返す、文字列処理ルーチンです。

構文

far-address = **StringAddress**(*string-descriptor%*);

string-descriptor%　Basic の文字列記述子の near アドレス。

解説

StringAddress は C 言語用の構文を持ちます。

複数言語によるプログラム以外では、StringAddress ルーチンは必要ありません。Basic だけで構成されたプログラムでは、可変長文字列の far アドレスを得るには **SSEGADD** 関数を使います。

StringAddress ルーチンの呼び出しは通常、Basic 以外の言語ルーチンから行います。これによって Basic 以外の言語ルーチンから Visual Basic へ転送した文字列のアドレスを調べることができます。

例えば、MASM のルーチンから Basic の文字列領域に対して、StringAssign を使って文字列を渡したとします。その文字列の記述子は、オフセット descriptor のアドレスにあります。MASM のルーチンは、次のようにして文字列データの far アドレスを得ることができます。

```
.model
extrn       stringaddress: far
.code
lea         ax, descriptor        ; 記述子のオフセット
push        ax
call        stringaddress
```

far アドレスは DX:AX に返されます。DX はセグメントを、AX はオフセットを表します。

参照

StringAssign ルーチン，StringLength ルーチン，StringRelease ルーチン

StringAssign ルーチン

機能

異なる言語のメモリ領域の間で文字列データを転送する、文字列処理ルーチンです。

構文

StringAssign(*sourceaddress&, sourcelength%, destaddress&, destlength%*);

sourceaddress&　転送元の文字列が可変長なら、文字列記述子を指す far ポインタ。固定長なら文字列の先頭を指す far ポインタ。

sourcelength%　転送元の文字列が可変長なら、0。固定長なら、その文字列のバイト単位の長さ。

destaddress&　転送先の文字列が可変長なら、文字列記述子を指す far ポインタ。固定長なら文字列の先頭を指す far ポインタ。

destlength%　転送先の文字列が可変長なら、0。固定長なら、その文字列のバイト単位の長さ。

解説

StringAssign の構文は C 言語用ですが、構文中の引数の順序は Basic, Pascal, FORTRAN での呼び出し規約に従っており、C 言語の呼び出し規約とは異なります(Basic, Pascal, FORTRAN では引数はソースコードに記述されるとおりの順序で渡され、C ではその逆になります)。

StringAssign ルーチンは、文字列を、ある言語のメモリ領域から他の言語の領域へ転送します。このルーチンは、Basic の可変長文字列を相手側言語の固定長文字列へ移すとき、あるいはその逆のときなどに使います。通常、StringAssign ルーチンは、Basic の文字列に対して処理を行う、Basic 以外の言語プログラムから呼び出されます。ただし Basic プログラムから呼び出すこともできます。

MASM, C, Pascal, FORTRAN では、固定長文字列しか使用できません。これらの言語でプログラミングするときには、StringAssign を使って新たな Basic の可変長文字列を作成し、そこに固定長文字列データを転送します。例えば "Hello" というデータを持つ MASM の固定長文字列を Basic の可変長文字列に転送するには、次のようにします。

```
fixedstring     db      "Hello"     ; 転送元のデータ
descriptor      dd       0          ; 転送先の記述子
```

2 番目のデータ要素である記述子は、0 で初期化された 4 バイトの文字列記述子です。このように文字列記述子が 0 で初期化されている場合、StringAssign は新たな可変長文字列を生成して、それに合わせて文字列記述子を設定します。

新たな可変長文字列を生成するとき、上の例のように文字列記述子に 4 バイトの静的データを確保する必要があります。スタックを割り当てても正しく動作しません。

新たな Basic の可変長文字列を生成し、そこに固定長データを転送するには、StringAssign は次の 4 個の引数を必要とします。

- MASM データ セグメントの固定長文字列の far アドレス
- MASM データ セグメントの固定長文字列の長さ
- MASM データ セグメントの文字列記述子の far アドレス
- 0 (Basic データ セグメントの文字列が可変長になることを表す)

次に示す MASM のコードは、これらの引数をスタックにプッシュして StringAssign を呼び出します。

```
.model
extrn       stringassign:  far
.code
push        ds                      ; 固定長文字列のセグメント
lea         ax, fixedstring         ; 固定長文字列のオフセット
push        ax
mov         ax, 5                   ; 文字列 "Hello" の長さ
push        ax
push        ds                      ; 記述子のセグメント
lea         ax, descriptor          ; 記述子のオフセット
push        ax
mov         ax, 0                   ; 0 で可変長文字列を指定
push        ax
call        stringassign
```

StringAssign が呼び出されると、Visual Basic は 4 バイトの descriptor に適切な文字列記述子を書き込みます。

参照　StringAddress ルーチン，StringLength ルーチン，StringRelease ルーチン

StringLength ルーチン

機能　Visual Basic の可変長文字列の長さを返す、文字列処理ルーチンです。

構文　*length* = **StringLength**(*string-descriptor%*);

string-descriptor%　Basic の文字列記述子の near アドレス。

解説　StringLength は C 言語用の構文を持ちます。

複数言語によるプログラム以外では、StringLength ルーチンは必要ありません。Basic だけで構成されたプログラムでは、可変長文字列の長さを得るには **LEN** 関数を使います。

StringLength ルーチンの呼び出しは通常、Basic 以外の言語ルーチンから行います。

例えば、MASM のルーチンがパラメータとして文字列記述子へのポインタを受け取るとします。MASM のルーチンは、次のようにして文字列データの長さを得ることができます。

```
.model
extrn       stringlength: far

.code
mov         ax, psdparm             ; 記述子のオフセット
push        ax
call        stringlength
```

文字列の長さは AX に返されます。

参照 LEN 関数，StringAddress ルーチン，StringAssign ルーチン，StringRelease ルーチン

StringRelease ルーチン

機能 StringAssign を使って、他の言語によって生成された可変長文字列データの領域を解放する、文字列処理ルーチンです。

構文 **StringRelease**(*string-descriptor%*);

string-descriptor% Basic 以外の言語ルーチンで割り当てた、文字列記述子の near アドレス。

解説 StringRelease は C 言語用の構文を持ちます。

StringRelease ルーチンは、Basic 以外の言語プログラムによって割り当てられた可変長文字列に対してだけ使ってください。Basic が生成した文字列に対して使った場合、結果は予測できません。

StringRelease ルーチンの呼び出しは、Visual Basic のデータ領域を解放するために、Basic 以外の言語ルーチンから行われます。

Visual Basic が生成した文字列は、すべて Visual Basic によって管理されており、解放も自動的に行われます。しかし、Basic 以外の言語ルーチンから Basic へ転送された文字列は、StringRelease ルーチンで明確に解放しない限り、そのまま残ります。

Basic 以外の言語ルーチンから Visual Basic に転送された文字列を解放するために StringRelease を使用する必要があるのは、StringAssign が文字列だけを転送して文字列記述子を転送しないためです。文字列記述子がないと Visual Basic は文字列を解放できません。

例えば、MASM のルーチンから Basic の文字列領域に対して、StringAssign を使って文字列が渡されたとします。その文字列の記述子は、オフセット descriptor のアドレスにあります。MASM のルーチンは、次のようにして文字列データを解放することができます。

```
.model
extrn       stringrelease: far

.code
lea         ax, descriptor          ; 記述子のオフセット
push        ax
call        stringrelease
```

参照 StringAddress ルーチン，StringAssign ルーチン，StringLength ルーチン

Style プロパティ

機能

コンボ ボックスの種類と、コンボ ボックスのリスト ボックス部分の動作を決定します。実行時には読み込み専用です。

構文

[*formname.*]*ctlname*.**Style**

ctlname コンボ ボックスの名前。

対象

コンボ ボックス

解説

コンボ ボックスには、それぞれ機能が異なる 3 種類のスタイルがあります。Style プロパティは、このコンボ ボックスの種類を決めます。デザイン時にフォーム デザイナで、0 (ドロップダウン コンボ), 1 (シンプル コンボ), 2 (ドロップダウン リスト) に設定できます。実行時には変更できません。デフォルトは 0 (ドロップダウン コンボ) です。

ドロップダウン コンボ (Style プロパティが 0) は、通常のコンボ ボックスです。右側の下向き矢印をクリックすると、リスト ボックス部分がドロップダウンします。ユーザーは、ドロップダウンしたリスト中から項目を選択することもできますし、テキスト ボックス部分に直接値を入力することもできます。フォーカスが別のオブジェクトに移ると、自動的にリスト ボックス部分はクローズします。次のような特長があります。

- Height プロパティの値は常に 1 です。
- Text プロパティの値は、テキスト ボックス部分に表示されている文字列を表します。
- リスト ボックス部分がドロップダウンするとき、DropDown イベントが発生します。
- SelLength, SelStart, SelText の各プロパティは有効です。
- テキスト ボックス部分のテキストが変更されると、Change イベントが発生します。

シンプル コンボ (Style プロパティが 1) は、リスト ボックス部分が常に表示されている点を除き、ドロップダウン コンボと同じ機能です。定形の項目が少ない場合に使います。次のような特長があります。

- リスト ボックス部分の行数は、Height プロパティで決まります。
- Text プロパティの値は、テキスト ボックス部分に表示されている文字列を表します。
- DropDown イベントは発生しません。
- SelLength, SelStart, SelText の各プロパティは有効です。
- テキスト ボックス部分のテキストが変更されると、Change イベントが発生します。

ドロップダウン リスト (Style プロパティが 2) は、右側の下向き矢印をクリックすると、リスト ボックス部分がドロップダウンする点では、ドロップダウン コンボと同じです。ただしユーザーは、リスト ボックス部分に直接値を入力することはできず、リストから項目を選択するだけです。選択できる項目が固定されている場合に使います。次のような特長があります。

- Height プロパティの値は常に 1 です。
- Text プロパティの値は、リスト ボックス部分で反転表示されている項目を示します。
- リスト ボックス部分がドロップダウンするとき、DropDown イベントが発生します。
- SelLength, SelStart, SelText の各プロパティは使用できません。
- Change イベントは発生しません。

参照

Change イベント，Click イベント，DblClick イベント，DropDown イベント，Height プロパティ，SelLength プロパティ，SelStart プロパティ，SelText プロパティ，Text プロパティ

SUB ステートメント

機能

SUB プロシージャの名前，引数とそのデータ型を宣言し、SUB プロシージャの始まりを示します。

構文

[STATIC]SUB *globalname* [(*parameterlist*)]

[*statementblock*]

[EXIT SUB]

[*statementblock*]

END SUB

STATIC
プロシージャ中のローカル変数の値が、次に呼び出されるまで保存されることを示します。省略すると、ローカル変数は、プロシージャが呼び出されるたびに値が代入され、プロシージャの実行が終了すると値が失われます。プロシージャの外 (モジュールレベル) で宣言された変数には影響しません。

name
定義する SUB プロシージャの名前。プログラム中ですでに使われている名前は指定できません。プロシージャ名は、 Visual Basic の変数の名前付け規約に従います。

parameterlist
呼び出し側との受け渡しに使う変数とそのデータ型を、カンマで区切って指定します。

statementblock
その SUB プロシージャの中で実行される、任意の行数のステートメント。

EXIT SUB SUB プロシージャを終了し、その SUB プロシージャを呼び出したステートメントの直後のステートメントに制御を戻します。1 つの SUB プロシージャ中に複数の **EXIT SUB** を記述することもできます。

parameterlist 引数には、次の構文に従って引数とそのデータ型を指定します。

[**BYVAL**]*variable*[()][**AS** *type*][, [**BYVAL**]*variable*[()][**AS** *type*]]...

BYVAL 直後に指定されている変数に対して、デフォルトの引き渡し方法である参照による引き渡しではなく、値による引き渡しで渡されることを示します。**BYVAL** は、数値の基本データ型 (**INTEGER, LONG, SINGLE, DOUBLE, CURRENCY**) と可変長文字列 (**STRING**) に対してだけ指定できます。

variable 渡された引数を受け取る変数の名前。配列変数の場合、名前の後ろに空のカッコ () を付けます。名前の最後に型宣言文字 (%, &, !, #, @, $) を付けてデータ型を指定することができます。この場合 **AS** *type* は省略します。ユーザー定義型に含まれるものでなければ、固定長文字列変数を指定することはできません。

type 変数のデータ型。変数それぞれに対して指定します。 **INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING**, ユーザー定義型, **FORM, CONTROL** のいずれかが指定できます。 **FORM** は引数としてフォームが渡されることを示し、 **CONTROL** はいずれかのコントロールが渡されることを示します。

解説

SUB と **END SUB** は、SUB プロシージャの始まりと終わりに記述します。また、プロシージャ内の任意の場所に、SUB プロシージャの実行を終了させる **EXIT SUB** ステートメントを置くことができます。

SUB プロシージャは、**FUNCTION** プロシージャと同様に、独立したプロシージャです。引数を受け取り、一連のステートメントを実行し、引数の値を変更します。ただし **FUNCTION** プロシージャとは異なり、SUB プロシージャは呼び出し側に値を返しません。

SUB プロシージャは、**CALL** ステートメントを使うか、SUB プロシージャ名の後に引数のリストを指定して呼び出します。また、その SUB プロシージャがイベント プロシージャなら、対応するイベントが発生したときに自動的に呼び出されます。

プロシージャ中で使われる変数は、**SUB** ステートメントで引数として指定された変数を除き、原則としてローカル変数として扱われます。ただし、**SHARED** ステートメントで共有変数として宣言された変数や、**DIM** または **COMMON** ステートメントで **SHARED** キーワードを指定して宣言された変数は、ローカル変数ではありません。

SUB プロシージャがフォーム モジュールに含まれているときは、そのフォーム モジュール中のどこからでも呼び出すことができます。(SUB プロシージャ名は、そのフォーム モジュールに対してはグローバルです)。ただし、他のモジュールから呼び出すことはできません。コード モジュールに含まれているときは、プログラム全体のどこからでも呼び出すことができます (プログラム全体に対してグローバルです)。したがって、同じ名前を変数や他のプロシージャの名前にすることはできません。

変数に対して **BYVAL** キーワードが指定されていなければ、実引数に変数が指定されると、参照によって引き渡されます。プロシージャ内でその変数の値を変更すると、呼び出し側の値も変更されます。実引数に式が指定されたときは、常に値で引き渡され、呼び出し側の変数には影響しません。**BYVAL** キーワードが指定されていなくても、プロシージャを呼び出すときに変数をカッコで囲んで指定すると、式と見なされて値で渡されます。

STATIC キーワードを使うと、プログラムの実行速度がわずかに上がります。ただし、メモリの使用効率が落ちる場合があります。

SUB プロシージャは再帰呼び出しできます。つまり、自分自身を呼び出せます。再帰的な SUB プロシージャでは、**STATIC** キーワードは指定しません。

Visual Basic では、文字列を SUB プロシージャに渡す前に、単純固定長文字列引数を可変長文字列引数に変換するため、SUB プロシージャを呼び出すときの引数 (実引数) として固定長文字列の配列を使うことはできません。

Basic は、演算の効率を高めるため、数式を並べ替えることがあります。このため、変数の値を変えてしまうような FUNCTION プロシージャは、数式の中で使わないでください。

入出力を行うステートメントから呼び出される SUB プロシージャでは、入出力ステートメントを使わないでください。使用すると結果は予測できません。

GOSUB, GOTO, RETURN によって、SUB プロシージャに入ったり、SUB プロシージャから抜けたりすることはできません。

参照　**CALL** ステートメント (Basic プロシージャ)，**DECLARE** ステートメント，**FUNCTION** ステートメント

SWAP ステートメント

機能　2 つの変数の値を交換します。

構文　**SWAP** *variable1*, *variable2*

variable1	同じデータ型の 2 つの変数。
variable2	

解説　引数には、ユーザー定義型を含むどのようなデータ型の変数でも指定できますが、2 つの変数のデータ型が一致していないと、"変数または式の型が一致しません" エラーが発生します。

参照　**LET** ステートメント，**LSET** ステートメント

System プロパティ

Archive プロパティを参照してください。

SYSTEM ステートメント

機能

オープンされているすべてのファイルをクローズし、オペレーティング システムに制御を戻すステートメントです。

構文

SYSTEM [*n*%]

n%	オペレーティング システムに返す値。省略すると、0 と見なされます。

解説

トラップされなかったエラーや致命的なエラーでプログラムの実行が中止されたときは、*n*% 引数の指定にかかわらず -1 が返されます。

Visual Basic 起動時に /RUN オプションで指定して実行したプログラムでは、**SYSTEM** ステートメントを実行すると、制御はオペレーティングシステムに戻ります。

イミディエイト ウィンドウで **SYSTEM** ステートメントを実行すると、Visual Basic は終了します。

/O オプションを付けてコンパイルした独立型の実行可能 (.EXE) ファイルでは、**SYSTEM** ステートメントはすべてのファイルをクローズし、オペレーティング システムへ戻ります。このとき *n*% を返します。プログラミング環境上では、プログラムの実行を停止し、ファイルをクローズしてプログラミング環境に戻ります。

参照

END ステートメント，**STOP** ステートメント，System プロパティ

TAB 関数

機能

PRINT, LPRINT, PRINT # ステートメントおよび **PRINT** メソッド中で使い、データの出力位置を移動する関数です。

構文

TAB(*column%*)

column%　　移動先の桁位置を示す 1～80 の整数。

解説

column% には、次に表示を行う桁位置を表す数式を指定します。左端を 1 と数え、右端は、出力先のデバイスの 1 行の桁数に相当します。

出力デバイスの桁数は **WIDTH** ステートメントを使って設定できます。

PRINT ステートメントで **TAB** 関数を使った例を次に示します。

```
PRINT  TAB(10); City$; TAB(40); State$; TAB(45); Zip$
```

TAB 関数の動作は次の 3 つの値に依存します。

- 引数 *column%*
- **TAB** 関数を実行したときの出力行上の出力位置
- そのときの出力行の桁数

n% にそのときの出力行の桁数以上の値を指定すると、桁数で割った余りと見なされます。つまり次のように計算されます。

TAB(*column%* **MOD** 桁数)

指定した桁位置が、現在の表示位置より前の場合、または *column%* に 1 行の桁数以上の値を指定した場合は、次の行に改行されます。*column%* に 1 未満の値を指定すると、1 と見なされます。

参照

LPRINT ステートメント，**PRINT** メソッド，**PRINT** ステートメント，**PRINT #** ステートメント，**SPACE$** 関数，**SPC** 関数

TabIndex プロパティ

機能

コントロールのタブ オーダーを決めます。

構文

[*formname*.]*ctlname*.**TabIndex**[= *index%*]

index% タブ オーダーを示す整数。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フレーム，水平スクロール バー，ラベル，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

TabIndex プロパティに設定される値は、0 以上で、“同じフォームで TabIndex プロパティを持つコントロールの数 -1” までの整数です。TabIndex プロパティの値は、コントロールをフォーム上に配置していくと、自動的に割り振られて行きます。また、あるコントロールの TabIndex プロパティの値を変更したり、コントロールを削除または追加した場合、自動的に他のコントロールの TabIndex プロパティの値が振り直されます。

フォーム上に配置されているコントロールまたは子フォームのいずれかがフォーカスを持つことができる場合は、フォーム自身はフォーカスを持つことはできません。一方ピクチャー ボックスは、子コントロールがあるときでもフォーカスを持つことができます。

参照

TabStop プロパティ

TabStop プロパティ

機能

[TAB] キーでフォーカスを移動するとき、そのコントロールにフォーカスが移るかどうかを決めます。

構文

[*formname*.]*ctlname*.**TabStop**[= *boolean%*]

boolean% [TAB] キーでフォーカスが移るかどうかを示す値。 True (-1) は移ることを示し、False (0) は移らないことを示します。デフォルトは True (-1) です。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，水平スクロール バー，リスト ボックス，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

TabStop プロパティは、ユーザーが [TAB] キーを使ってフォーカスを移動するときに、そのコントロールにフォーカスが移るかどうかだけを決めます。TabStop プロパティが False (0) のコントロールも、マウスでクリックすればフォーカスを持たせることができます。また、TabStop プロパティの値を変更しても、TabIndex プロパティで設定したタブ オーダーは影響されません。

オプション ボタンを使うときは、各オプション ボタンそれぞれにフォーカスが移動していくのではなく、オプション ボタンのグループに対して 1 回だけフォーカスが移るようにした方が便利です。この場合、グループの最初のオプション ボタンの TabStop プロパティを True (-1) に設定し、他のオプション ボタンは False (0) にします。

フォーム上に配置されているコントロールまたは子フォームのいずれかがフォーカスを持つことができる場合は、フォーム自身はフォーカスを持つことはできません。一方ピクチャー ボックスは、子コントロールがあるときでもフォーカスを持つことができます。

参照

TabIndex プロパティ

Tag プロパティ

機能

各フォームまたはコントロールを判別するための、各オブジェクトに固有の名前を設定または取得します。

構文

[*formname.*][*ctlname.*]**Tag**[= *string-expression$*]

string-expression$ オブジェクトを判別するためのタグを示す文字列式。デフォルトは null 文字列 ("") です。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，テキスト ボックス，タイマー，垂直スクロール バー

解説

Tag プロパティは、フォームやコントロールを引数としてプロシージャに渡すときなど、どのフォームまたはコントロールであるかを判別する必要があるときに使います。Tag プロパティは、そのオブジェクト自身の動作や他のオブジェクトにはいっさい影響せず、単なる判別用の文字列の意味だけを持ちます。

フォームやコントロールに対する識別子としては FormName または CtlName プロパティの値が使われますが、これらのプロパティの設定値をコード中で参照することはできません。また、コントロールの種類は **IF TYPEOF** ステートメントで調べることができますが、同じ種類のコントロールが複数ある場合、個々のコントロールの判別はできません。

Tag プロパティは、他のプロパティでは扱えない情報をメモとして保持するためにも使えます。

参照　CtlName プロパティ，FormName プロパティ

TAN 関数

COS 関数を参照してください。

Text プロパティ

機能　テキスト ボックスやコンボ ボックスのテキスト ボックス部分に表示されているテキストや、リスト ボックスやコンボ ボックスのリスト ボックス部分で選択されている文字列を、取得または設定します。リスト ボックスと Style プロパティが 2 に設定されているコンボ ボックスでは、実行時には読み込み専用です。

構文　[*formname.*]*ctlname.***Text**[= *string-expression$*]

ctlname	テキスト ボックス，コンボ ボックス，リスト ボックスの名前。
string-expression$	表示されるテキストを示す文字列式 (テキスト ボックスと Style プロパティが 0 または 1 に設定されているコンボ ボックスのみ)。デフォルトでは、CtlName プロパティのデフォルト値 (デフォルトのコントロール名) と同じです。

対象　コンボ ボックス，リスト ボックス，テキスト ボックス

解説　Text プロパティの意味は、コントロールによって 2 種類に分けられます。

テキスト ボックスと、Style プロパティが 0 (ﾄﾞﾛｯﾌﾟﾀﾞｳﾝｺﾝﾎﾞ) または 1 (ｼﾝﾌﾟﾙｺﾝﾎﾞ) に設定されているコンボ ボックスでは、Text プロパティの値は、テキスト ボックスまたはコンボ ボックスのテキスト ボックス部分に表示される文字列を示します。最大 32KB の文字列が設定できます。デフォルトでは、CtlName プロパティ (コントロール名) のデフォルト値と同じです。

リスト ボックスと、Style プロパティが 2 (ﾄﾞﾛｯﾌﾟﾀﾞｳﾝﾘｽﾄ) に設定されているコンボ ボックスでは、Text プロパティの値は、リスト ボックス部分で反転表示されている項目のテキストを示します。この場合、Text プロパティはデザイン時には操作できず、実行時には読み込み専用です。つまりコード中から値を得ることはできますが、設定はできません。反転表示されている項目がなければ、null 文字列 ("") が返されます。次の 2 つは同じ意味です。

```
ctlname.Text
ctlname.List(ctlname.ListIndex)
```

参照　Caption プロパティ，SelLength プロパティ，SelStart プロパティ，SelText プロパティ

TEXTHEIGHT, TEXTWIDTH メソッド

機能

TEXTHEIGHT はテキスト文字列の高さを行数で返します。**TEXTWIDTH** はテキスト文字列の長さをバイト数で返します。

構文

[{[*formname.*]*ctlname.*|*formname.*}]**TEXTHEIGHT**(*string-expression$*)

[{[*formname.*]*ctlname.*|*formname.*}]**TEXTWIDTH**(*string-expression$*)

ctlname	ピクチャー ボックスの名前。
string-expression$	高さや長さを求める対象となるテキスト文字列。

対象

フォーム，ピクチャー ボックス

解説

TEXTHEIGHT と **TEXTWIDTH** メソッドは、関数のように値を返します。次のように数式の中で使うことに注意してください。

```
A = TEXTWIDTH ("Test String")
```

TEXTHEIGHT メソッドは、*string-expression$* で指定した文字列の行数、つまりその文字列に含まれている CR+LF (キャリッジリターン + ラインフィード) の数に 1 を加えたものを返します。

TEXTWIDTH メソッドは、*string-expression$* で指定した文字列の長さをバイト数で返します。その文字列に CR が含まれているときは、最長の行のバイト数を返します。CR が含まれていないときは、返される値は **LEN** 関数と同じです。

適用するオブジェクトがフォームであるかピクチャー ボックスであるかにかかわらず、返される値は同じです。

参照

LEN 関数，**KLEN** 関数

TIME$ 関数, TIME$ ステートメント

機能

TIME$** 関数は引数を取らず、現在の時刻を表す文字列を返します。**TIME$ ステートメントは、現在の時刻を設定します。

構文

**TIME$**[= *string-expression$*]

string-expression$　設定する時刻。

解説

時刻は、次の形式で表されます。時刻は 24 時間制です。

hh:mm:ss

hh は時を表します (00～23)。mm は分を表します (00～59)。ss は秒を表します (00～59)。

参照

FORMAT$ 関数

Timer イベント

機能

あらかじめ設定されたタイマー コントロールの時間間隔が経過したときに発生します。

構文

SUB *ctlname*_**Timer** ([*Index* **AS INTEGER**])

Index　コントロール配列のインデックス番号が渡されます (*ctlname* がコントロール配列のときのみ)。

対象

タイマー

解説

Timer イベントを使えば、一定の時間間隔ごとに処理を行うことができます。

Timer イベントが発生する間隔は、タイマー コントロールの Interval プロパティで設定します。単位はミリ秒です。例えば Interval プロパティを 1,000 に設定すると、Timer イベントは 1 秒に一回実行されます。ただし実際に検出できる時間間隔は、ハードウェアに依存します。

Timer イベントを発生させるには、タイマー コントロールの Enabled プロパティを True (-1) に設定し、さらに Interval プロパティを 0 以外の値に設定しておく必要があります。

参照

Enabled プロパティ，Interval プロパティ，**ON TIMER** ステートメント，**TIMER** 関数，**TIMER** ステートメント

TIMER ステートメント

機能

タイマー イベントのトラップを有効，無効，中断状態にします。フォームやコントロールに関連したイベントには影響を与えません。フォームを使うアプリケーションでは、このステートメントではなく、タイマー コントロールを使ってください。

構文

TIMER {ON|OFF|STOP}

ON トラップを有効にすることを示します。 **ON TIMER** ステートメントによる、タイマー イベントのトラップが可能になります。

OFF トラップを無効にすることを示します。このステートメントが実行されたあとに起こったイベントは、記憶されません。

STOP トラップを中断します。このステートメントが実行されたあとでタイマー イベントが発生すると、そのイベントは記憶されます。次に **TIMER ON** ステートメントが実行されると、すぐに **ON TIMER** ステートメントでトラップされます。

解説

このステートメントで扱うイベントは、フォームやコントロールを使ったイベント ドリブン プログラミング モデルのイベントとは異なることに注意してください。フォームやコントロールを使うアプリケーションでは、タイマー コントロールを使う方が簡単です。ただし、フォームやコントロールを使ったアプリケーションでも、**TIMER** ステートメントを使うことができます。

イベント トラップが有効であるとき、そのイベントが発生していないかどうか (設定されている秒数が経過していないか) が自動的に監視されています。このチェックは、プログラミング環境で実行されている場合と、/V オプション付きでコンパイルされた実行可能ファイルでは、ステートメントごとに行われます。/W オプション付きでコンパイルされた実行可能ファイルでは、行番号または行ラベルごとにチェックされます。イベントが発生していると、**ON TIMER** ステートメントで設定されているイベント処理ルーチンに制御が移ります。

イベント トラップを行うステートメントを含むプログラムを BC コマンドラインからコンパイルするとき、/W または /V オプションを付けずにコンパイルすると、"ON event ステートメントを含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

TIMER STOP ステートメントでトラップを中断している間に起きたイベントは、**TIMER OFF** ステートメントが実行されると失われます。

イベントがトラップされ、対応する処理ルーチンに制御が移ると、再帰的なトラップが起こらないように自動的に **TIMER STOP** が実行されます。処理ルーチンが終了すると、そのルーチンの中で明確に **TIMER OFF** が実行されていない限り、自動的に **TIMER ON** ステートメントが実行されます。

参照

ON TIMER ステートメント，**TIMER** 関数

TIMER 関数

機能　午前 0 時から経過した秒数を返す関数です。

構文　**TIMER**

参照　**ON TIMER** ステートメント，**RANDOMIZE** ステートメント，**TIMER** ステートメント

TIMESERIAL 関数

機能　数値で指定された時刻を表す、シリアル値を返します。

構文　**TIMESERIAL**(*hour%*, *minute%*, *second%*)

hour%	時 (0 (12:00 A.M.) ～23 (11:00 P.M.))。
minute%	分 (0～59)。
second%	秒 (0～59)。

解説　結果として得られるシリアル値が正の数である限り、引数に負の数を指定できます。例えば、2:00:02 の 50 秒前のシリアル値を得るには、次のようにすることができます。

```
TIMESERIAL(2, 0, 2-50)
```

TIMEVALUE 関数

機能　文字列で指定された時刻を表す、シリアル値を返します。

構文　**TIMEVALUE**(*time$*)

time$	0:00:00 (12:00:00 A.M.) ～23:59:59 (11:59:59 P.M.) までの、任意の有効な時刻を示す文字列。"2:24PM" や "14:24" のように指定できます。*time$* に日付の情報が含まれていても無視されます。

Top プロパティ

Left プロパティを参照してください。

TRON, TROFF ステートメント

機能

プログラム ステートメントの実行をトレースする、デバッグのためのステートメントです。フォームが表示されている状態では利用できません。

構文

TRON

TROFF

解説

TRON ステートメントはトレースを有効にし、**TROFF** ステートメントは無効にします。

フォームが表示されているときに **TRON** または **TROFF** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

プログラミング環境では、**TRON** ステートメントは [ﾃﾞﾊﾞｯｸﾞ] メニューの [トレース ON/OFF] コマンドと同じ機能です。いずれも、実行中のステートメントが強調表示されます。

コンパイルして作成した実行可能ファイルで **TRON** および **TROFF** ステートメントを使うとき、行番号が表示されるようにするには次のようにします。

- [実行] メニューの [EXE ファイル作成] を選択し、[EXE ファイル作成] ダイアログボックスで [ランタイム エラー チェック] オプションをオンにする。
- BC コマンドラインで、/D オプションを指定する。

TYPE ステートメント

機能

ユーザー定義のデータ型を宣言します。

構文

TYPE *usertype*

elementname **AS** *typename*

[*elementname* **AS** *typename*]

.
.
.

END TYPE

usertype	ユーザー定義のデータ型の名前。ユーザー定義型の名前は、Visual Basic の変数の名前付け規約に従います。
elementname	ユーザー定義のデータ型の要素。配列を指定するときは、各次元の添字の最小値と最大値を定数で指定し、静的配列にしなければなりません。動的配列をユーザー定義型に含めることはできません。
typename	*elementname* で指定した要素のデータ型。**INTEGER, LONG, SINGLE, DOUBLE, CURRENCY, STRING** **n*, 別のユーザー定義型が指定できます。可変長文字列は指定できません。 ユーザー定義型の変数は構造体とも呼ばれます。

解説

ユーザー定義型は、プログラムで使う前に、あらかじめ **TYPE** ステートメントで宣言しなければなりません。**TYPE** ステートメントは、必ずモジュール レベルに記述してください。変数をユーザー定義型として宣言するときは、**DIM, REDIM, COMMON, STATIC, SHARED** などのステートメントで、そのユーザー定義型の名前を指定します。一度宣言すれば、ユーザー定義型の変数は、基本データ型の変数と同じように使うことができます。

ユーザー定義型の中に文字列を含めるときは、固定長文字列でなければなりません。例えば、ユーザー定義型 sampletype に、40 バイトの固定長文字列 Keyword を含めるときは、次のように指定します。

```
TYPE sampletype
    Keyword AS STRING * 40
    .
    .
    .
END TYPE
```

また、ユーザー定義型の中に配列を含めることもできます。この配列は必ず静的配列になります。配列を指定するときは、各次元の添字の最大値と最小値を、定数で指定してください。例えばユーザー定義型 StateData の中に、100 個の要素を持つ静的配列 CityCode を含めるときは、次のように指定します。

```
TYPE StateData
    CityCode(1 TO 100) AS INTEGER
    County AS STRING * 30
END TYPE
DIM Washington(1 TO 100) AS StateData
```

効率を高めるために、ユーザー定義型の中で使われる配列は偶数オフセットから始まるようにしてください。

ユーザー定義型の変数は、1つで多くの値を格納できるため、それだけサイズが大きくなります。特にユーザー定義型中に別のユーザー定義型や静的配列を含めると、非常に大きなデータ型が作成できます。このような変数をプロシージャとの間で受け渡しを行うと、それだけスタック領域を使うため、スタック領域が不足する原因になります。

参照　**DIM** ステートメント，**REDIM** ステートメント，**COMMON** ステートメント，**STATIC** ステートメント，**SHARED** ステートメント

UBOUND 関数

LBOUND 関数を参照してください。

UCASE$ 関数

LCASE$ 関数を参照してください。

UEVENT ステートメント

機能

ユーザー定義イベントのトラップを有効，無効，中断状態にします。フォームやコントロールに関連したイベントには影響を与えません。

構文

UEVENT {ON|OFF|STOP}

ON トラップを有効にすることを示します。 **ON UEVENT** ステートメントによる、ユーザー定義イベントのトラップが可能になります。

OFF トラップを無効にすることを示します。このステートメントが実行されたあとに起こったイベントは、記憶されません。

STOP トラップを中断します。このステートメントが実行されたあとでユーザー定義イベントが発生すると、そのイベントは記憶されます。次に **UEVENT ON** ステートメントが実行されると、すぐに **ON UEVENT** ステートメントでトラップされます。

解説

このステートメントで扱うイベントは、フォームやコントロールを使ったイベント ドリブン プログラミング モデルのイベントとは異なることに注意してください。フォームやコントロールに関連したイベントには影響を与えません。ただし、フォームやコントロールを使ったアプリケーションでも使うことができます。

イベント トラップが有効であるとき、そのイベントが発生していないかどうか (ユーザー定義イベントが起こっていないか) が自動的に監視されています。このチェックは、プログラミング環境で実行されている場合と、/V オプション付きでコンパイルされた実行可能ファイルでは、ステートメントごとに行われます。/W オプション付きでコンパイルされた実行可能ファイルでは、行番号または行ラベルごとにチェックされます。イベントが発生していると、**ON UEVENT** ステートメントで設定されているイベント処理ルーチンに制御が移ります。

イベント トラップを行うステートメントを含むプログラムを BC コマンドラインからコンパイルするとき、/W または /V オプションを付けずにコンパイルすると、"ON event ステートメントを含むプログラムは /V または /W を指定してコンパイルしてください" エラーが発生します。プログラミング環境の [EXE ファイル作成] コマンドで実行可能ファイルを作成するときは、これらのオプションは自動的に指定されます。

UEVENT STOP ステートメントでトラップを中断している間に起きたイベントは、**UEVENT OFF** ステートメントが実行されると失われます。

イベントがトラップされ、対応する処理ルーチンに制御が移ると、再帰的なトラップが起こらないように自動的に **UEVENT STOP** が実行されます。処理ルーチンが終了すると、そのルーチンの中で明確に **UEVENT OFF** が実行されていない限り、自動的に **UEVENT ON** ステートメントが実行されます。

参照

EVENT ステートメント，**ON UEVENT** ステートメント，SetUEvent ルーチン

Unload イベント

機能

フォームがメモリからアンロードされるときに発生します。

構文

SUB Form_Unload (*Cancel* **AS INTEGER**)

Cancel	キャンセル フラグ。

対象

フォーム

解説

Unload イベントは、フォームをクローズするときの確認処理を行ったり、フォームに含まれているデータをセーブするのに便利です。

フォームを再ロードすると、そのフォームのすべてのコントロールの内容は、デザイン時の設定に初期化されます。

Cancel 引数には、常に 0 が渡されます。この値を 0 以外に変更すると、フォームはアンロードされません。フォームの確認処理の結果、フォームをアンロードしたくないときなどに使います。

参照

Load イベント，**LOAD** ステートメント，**UNLOAD** ステートメント

UNLOAD ステートメント

機能

フォームをメモリから削除したり、コントロール配列からコントロールを削除したりします。

構文

UNLOAD {*formname*|*ctlname*[*index%*]}

formname	アンロードするフォームの名前。
ctlname	要素を削除するコントロール配列。
index%	コントロール配列中の位置を表す整数。

解説

フォームやコントロールをアンロードするとメモリから解放され、それまでのプロパティの設定は失われます。

フォームがアンロードされると、実行時にそのフォーム上に置かれたコントロールにはアクセスできなくなります。デザイン時にそのフォーム上に置かれたコントロールはそのまま残りますが、実行時にこれらのコントロールやそのプロパティに対して加えられた変更は失われます。また、フォームのプロパティに対して加えられた変更もすべて失われます。

モーダルなフォームがアンロードされると、プログラムの実行は、そのモーダル フォームを表示するために **SHOW** メソッドが使われたコード中の位置に戻ります。

アンロードされた、またはまだロードされていないフォーム上のコントロールへの参照を行うと、そのフォームは再ロードされます。

実行時にフォームに加えられたコントロール配列要素は、そのコントロールに対して **UNLOAD** ステートメントを使わなければ削除できません。アンロードされたコントロールのプロパティは、そのコントロールが再ロードされるときに初期化されます。

参照　**LOAD** ステートメント，Unload イベント

UNLOCK ステートメント

LOCK ステートメントを参照してください。

VAL 関数

STR$ 関数を参照してください。

Value プロパティ

機能　コントロールのカレント状態 (チェック ボックスまたはオプション ボタン)，ボタンが押されているかどうか (コマンド ボタン)，コントロール上のカレントの位置 (スクロール バー) を取得または設定します。

構文　[*formname.*]*ctlname.***Value**[= *setting%*]

ctlname　チェック ボックス，オプション ボタン，コマンド ボタン，水平スクロール バー，垂直スクロール バーの名前。

setting%　コントロールによって設定できる値が異なります。

対象　チェック ボックス，コマンド ボタン，水平スクロール バー，オプション ボタン，垂直スクロール バー

解説　*settings%* には、次に示す値が指定できます。ただしコマンド ボタンの Value プロパティはデザイン時には設定できません。また、Value プロパティを参照すると、次に示す値が返されます。

コントロール	値	説明
チェック ボックス	0	(デフォルト) 未チェック
	1	チェック
	2	無効
コマンド ボタン	False (0)	(デフォルト) 押されていない
	True (-1)	押されている。コード中からこのプロパティを設定すると、そのボタンの Click イベントが発生します。
オプション ボタン	False (0)	(デフォルト) 選択されていない
	True (-1)	選択されている
スクロール バー	Min (-32768) ～ Max (32767)	Min プロパティの設定値がデフォルト

参照　Max プロパティ，Min プロパティ

VARPTR$ 関数

機能　**DRAW** および **PLAY** ステートメントの中で使い、変数のアドレスを文字列の形で返すメモリ管理関数です。

構文　**VARPTR$**(*variablename*)

variablename　**DRAW** および **PLAY** ステートメントの命令を含む文字列変数。配列の要素であるとき、その配列は、**VARPTR$** 関数を使う前に宣言されていなければなりません。また、この配列は可変長文字列配列でなければなりません。

解説　正しい結果を得るために、**VARPTR$** 関数の戻り値は、関数を実行した直後に使ってください。

参照　**DRAW** ステートメント，**PLAY** ステートメント (ミュージック)，**VARPTR** 関数

VARPTR, VARSEG 関数

機能

数値変数のアドレス、または文字列変数の文字列記述子のアドレスを返す関数です。

構文

VARPTR(*variablename*)

VARSEG(*variablename*)

variablename　任意の変数。ユーザー定義型変数や、その要素でも構いません。

解説

VARPTR 関数はオフセット アドレスを返し、**VARSEG** 関数はセグメント アドレスを返します。

variablename に数値変数を指定した場合は、その変数の格納位置そのもののアドレスを返します。

可変長文字列変数を指定することは意味がありません。**VARSEG** および **VARPTR** 関数に可変長文字列変数を指定すると、可変長文字列変数そのものの格納位置ではなく、文字列記述子の格納位置を返します。文字列記述子には、その文字列の長さや、実際の文字列データのアドレスが格納されています。Visual Basic の文字列はすべて far 文字列であり、以前のバージョンの Basic と文字列記述子のフォーマットが異なります。Basic プログラムで可変長文字列変数のアドレスを調べるには、**SSEG** および **SADD**、または **SSEGADD** 関数を使ってください。また、Basic 以外の言語で書いたルーチンで文字列変数のアドレスを調べるには、StringAddress ルーチンを使ってください。

variablename に、前に定義されていない変数を指定すると、その変数が作成されて、アドレスが返されます。

Basic は、プログラム実行中に変数の格納位置を移動することがあるため、**VARPTR** および **VARSEG** 関数は、その戻り値が必要になる直前で実行するようにしてください。

VARPTR および **VARSEG** 関数は、**BLOAD** や **BSAVE** ステートメント、Absolute や Interrupt ルーチン、**PEEK** 関数、**POKE** ステートメントなどと組み合わせて使います。また、他の言語で書かれたプロシージャに配列を引き渡すときにも使います。

配列変数のアドレスを調べるときには、*variablename* に、配列の最初の要素を指定してください。

```
DIM A(150)
.
.
.
ArrAddress=VARPTR(A(1))
```

DGROUP に格納されている変数のアドレスを調べるには、**VARPTR** 関数だけを使います。far メモリに格納されている変数のアドレスを調べるときには、**VARSEG** 関数も必要です。

ファイルのバッファのアドレスを調べるために **VARPTR** を使用することはできません。ファイルに関する情報を調べるには **FILEATTR** 関数を使ってください。

EMS メモリ中の配列を Basic 以外の言語で書かれたプロシージャに渡さないでください。/Ea オプションを指定して Visual Basic (VBDOS.EXE) を起動した場合、次の配列は EMS メモリに格納されることがあります。

- 512 バイト～ 16K バイトの数値配列
- 512 バイト～ 16K バイトの固定長文字配列
- 512 バイト～ 16K バイトのユーザー定義型配列

Basic 以外の言語で書かれたプロシージャに配列を渡したいときは、/Ea オプションを指定せずに Visual Basic (VBDOS.EXE) を起動してください。この場合、配列は EMS メモリには格納されません。

参照　**SADD** 関数，**SSEG** 関数，**VARPTR$** 関数

VARSEG 関数

VARPTR 関数を参照してください

VIEW PRINT ステートメント

機能　テキスト ビューポートの境界を設定します。フォームが表示されている状態では利用できません。

構文　**VIEW PRINT** [*toprow%* **TO** *bottomrow%*]

toprow%	新しく設定するテキスト ビューポートの上端の位置を示す整数。
bottomrow%	新しく設定するテキスト ビューポートの下端の位置を示す整数。

解説　フォームが表示されているときに **VIEW PRINT** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

引数 *toprow%* と *bottomrow%* を省略すると **VIEW PRINT** ステートメントは画面全体をテキスト ビューポートとして初期化します。

参照　**VIEW** ステートメント

VIEW ステートメント

機能

グラフィックス ビューポートを定義します。フォームが表示されている状態では利用できません。

構文

VIEW [[**SCREEN**] (*x1!,y1!*)-(*x2!,y2!*) [,[*color%*] [,*border%*]]]

SCREEN	以降のグラフィックス ステートメントで指定する座標は、グラフィックス ビューポートに対する相対座標ではなく、画面に対する絶対座標であることを示します。この場合も、図形はビューポートの内部にだけ描かれます。省略すると、指定する座標はビューポートに対して相対的な座標であると見なされます。つまり、指定した座標に *x1!* と *y1!* が加えられます。
(*x1!,y1!*)-(*x2!,y2!*)	設定するビューポートの対角の座標。2 つの点は、現在のスクリーン モードによって決まる画面の座標の、内側でなければなりません。
color%	ビューポートを塗りつぶす色のパレット番号。省略すると塗りつぶされず、画面の背景色と同じになります。
border&	ビューポートの境界線の色を示すパレット番号。省略すると、境界線は引かれません。

解説

フォームが表示されているときに **VIEW** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

テキスト モードで **VIEW** ステートメントを実行しようとすると、“引数が有効範囲にありません” エラーが発生します。

VIEW ステートメントは、グラフィックス ビューポートまたはクリッピング領域と呼ばれる、画面上の領域を定義します。グラフィックス ステートメントの出力は、この領域の中に描かれます。

(*x1!,y1!*) および (*x2!,y2!*) が指す位置は、そのときのスクリーン モードで、画面の内部になければなりません。

すべての引数を省略すると、画面全体がビューポートになります。**RUN** および **SCREEN** ステートメントを実行すると、ビューポートは自動的に画面全体に再定義されます。

参照

VIEW PRINT ステートメント

Visible プロパティ

機能

フォームやコントロールが画面上に表示されるかどうかを決めます。

構文

[*formname.*][*ctlname.*]**Visible**[= *boolean%*]

boolean% 表示されるかどうかを示す値。True (-1) は表示されることを示し、False (0) はされないことを示します。デフォルトは True (-1) です。

対象

チェック ボックス，コンボ ボックス，コマンド ボタン，ディレクトリ リスト ボックス，ドライブ リスト ボックス，ファイル リスト ボックス，フォーム，フレーム，水平スクロール バー，ラベル，リスト ボックス，メニュー，オプション ボタン，ピクチャー ボックス，テキスト ボックス，垂直スクロール バー

解説

フォームやコントロールがロードされているが表示されていないとき、ユーザーの操作によってイベントは発生しません。ただしフォームやコントロールは、コード中からの操作によって発生したイベントは認識します。また Timer イベントは、そのタイマー コントロールを含むフォームがロードされていれば常に発生します。

プログラムの起動時にコントロールを表示しないようにするには、デザイン時またはフォームの Load イベントに対するイベント プロシージャで、そのコントロールの Visible プロパティを False に設定します。

コード中で Visible プロパティの値を変更すると、例えばあるコントロールをクリックすることによって、別のコントロールの表示/非表示を切り換えるような処理が可能です。

一般に、コントロールの表示/非表示の切り換えには Visible プロパティを使い、フォームには **SHOW** と **HIDE** メソッドを使います。ただし、フォームに対して **SHOW** および **HIDE** メソッドを使うのは、そのフォームの Visible プロパティをコード内で True (-1) または False (0) に設定することとまったく同じ機能です。

参照

HIDE メソッド，**SHOW** メソッド

WAIT ステートメント

機能　プログラムの実行を一時中断し、入力ポートの状態を調べます。

構文　**WAIT** *portnumber%*, *and-expression%*[, *xor-expression%*]

portnumber%	ハードウェアの I/O ポートを示す、0～65535 の整数。
and-expression%	ポートから入力したデータと AND 演算を行う整数式。演算結果が 0 でないときだけ、次のステートメントに制御が移ります。
xor-expression%	ポートから入力したデータと XOR 演算を行う整数式。演算結果が 0 でないときだけ、次のステートメントに制御が移ります。

解説　**WAIT** ステートメントは **INP** 関数を拡張したもので、特定のビット パターンが入力ポートから入力されるまで、プログラムの実行を中断します。次のようなチェックが行われます。

1. ポートから読み込まれたデータは、*xor-expression* で指定された値と XOR 演算されます。*xor-expression* が省略されているときは、0 と見なされます。
2. 結果の値は、*and-expression* に指定された値と AND 演算されます。
3. 結果の値が 0 の場合は、1 と 2 が繰り返されます。結果が 0 でなければ、次のステートメントに制御が移ります。

入力ポートからの値が、結果が 0 以外になるビット パターンを作ることができないと、無限ループに入ってしまうことがあります。このときは、コンピュータをリセットしなければなりません。

次の例では、MS-DOS の I/O ポートである HandShakePort から受け取ったビット パターンと、2 によって表されるビットパターン (00000010) との間で AND 演算が行われます。

```
WAIT HandShakePort, 2
```

参照　**INP** 関数，**ON COM** ステートメント，**OUT** ステートメント，**SLEEP** ステートメント

WEEKDAY 関数

機能　日付と時刻のシリアル値を評価して、週の何日目かを返します。

構文　**WEEKDAY**(*serial*)

serial	日付と時刻を表すシリアル値 (-53,688～65,380)。

WHILE...WEND ステートメント

機能

指定した条件が真である間、一連のステートメントの実行を繰り返すフロー制御ステートメントです。

構文

WHILE *condition*

statementblock

WEND

condition	ループの続行条件を示す、真 (0 以外の値) または偽 (0) を返す式。
statementblock	ループ内で実行される、任意の行数のステートメント。

解説

condition が真 (0 以外の値) の場合、**WEND** までのステートメント (*statementblock*) が実行されます。その後 **WHILE** に戻り、再び *condition* が評価されます。*condition* がまだ真であれば、再度 **WEND** までのステートメントが実行されます。*condition* が偽 (0) の場合は、制御は **WEND** の次のステートメントに移ります。

WHILE...WEND ループはネストできます。つまり、**WHILE...WEND** ループの内部に別の **WHILE...WEND** ループを入れることができます。この場合、**WEND** は最後に実行された **WHILE** に対応します。**WHILE** と **WEND** が 1 対 1 に対応していないと、"WEND がありません" または "WHILE がありません" エラーが発生します。

WHILE を実行せずに **WHILE...WEND** ループ中に制御を移すことは避けてください。原因の突き止めにくい、実行時エラーやプログラムの論理的なエラーを引き起こします。

DO...LOOP ステートメントを使うと、より強力で柔軟なループ制御構造を記述することができます。通常は **DO...LOOP** ステートメントを使います。

参照

DO...LOOP ステートメント，**FOR...NEXT** ステートメント

Width プロパティ

Height プロパティを参照してください。

WIDTH ステートメント

機能

ファイルやデバイスに出力する行の幅を割り当てます。また、画面の表示桁数や行数を設定します。

構文

WIDTH {*#filenumber%*|*device$*}, *width%*

WIDTH LPRINT *width%*

WIDTH {*screenwidth%*|,*screenheight%*|*screenwidth%, screenheight%*}

filenumber%	オープンしているデバイスのファイル番号。
device$	Basic がサポートしているデバイスを示す文字列。"CONS:" などが指定できます。
width%	1 行の桁数。
screenwidth%	画面の桁数。
screenheight%	画面の行数。

解説

フォームが表示されているときに SCRN: または CONS: に対して **WIDTH** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

WIDTH *#filenumber%, width%*

- LPT1: や CONS: など、すでにファイルとしてオープンしてある出力デバイスの桁数を設定します。
- *filenumber%* 引数は **OPEN** ステートメントで指定したファイル番号です。
- *filenumber%* の前には必ずシャープ記号 (#) を付けます。
- 実行すると、すぐに桁数が変更されます。

WIDTH *device$, width%*

- *device$* (デバイス ファイル名) で指定したデバイスの桁数を *width%* に設定します。デバイスは "LPT1:" のように文字列で指定します。
- 指定した桁数は、そのデバイスに対して次に **OPEN** ステートメントを実行したときに有効になります。
- このデバイスファイル名を指定する方法は、すでにオープンしてあるデバイスに対しては無効です。

WIDTH LPRINT *width%*

- 以後の **LPRINT** や **LPRINT USING** ステートメントで使う、プリンタの桁数を設定します。
- 次のステートメントと同じ意味を持ちます。

  WIDTH "LPT1:", *width%*

WIDTH {*screenwidth%*|, *screenheight%*|*screenwidth%*, *screenheight%*}

- 画面にテキストを表示するときの行数と桁数を設定します。
- *screenwidth%* には 80 だけを指定できます。

screenheight% に指定できる値は、機種によって異なります。『機種依存情報』を参照してください。

参照　**LPRINT** ステートメント，**PRINT** ステートメント，**SCREEN** ステートメント，**VIEW PRINT** ステートメント，Width プロパティ

WINDOW ステートメント

機能　ビューポート座標を定義します。フォームが表示されている状態では利用できません。

構文　**WINDOW**[[**SCREEN**](*x1!*,*y1!*)-(*x2!*,*y2!*)]

SCREEN	y 座標が、上から下に向かって増加することを示します。ビューポート座標の原点は、ビューポートの左上隅になります。省略すると、下から上に向かって増加します。ビューポート座標の原点は、ビューポートの左下隅になります。
(*x1!*,*y1!*)	ビューポートの左上角に割り当てる新しいビューポート座標を示す単精度浮動小数点数。
(*x2!*,*y2!*)	ビューポートの右下角に割り当てる新しいビューポート座標を示す単精度浮動小数点数。

解説　フォームが表示されているときに **WINDOW** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

WINDOW ステートメントは、ビューポート内の座標を示す、ビューポート座標を設定します。以降のグラフィックス ステートメントでは、このビューポート座標を使います。ビューポート座標は物理座標と異なり、単精度浮動小数点数で指定でき、座標のきざみ幅も自由に設定できるため、グラフィックスを描画するときに便利です。**SCREEN** キーワードを指定しないときは、絶対座標や物理座標と異なり、y 座標は下から上に向かって増加することに注意してください。

すべての引数を省略すると、ビューポート座標は、画面の物理座標と同じものになります。また、**RUN** ステートメントを実行しても、物理座標と同じになります。

ビューポートのサイズを変更するには **VIEW** ステートメントを使います。**WINDOW** ステートメントを実行する前に、**SCREEN** キーワードを指定して **VIEW** ステートメントを実行しているときは、ビューポート座標はビューポート内ではなく、画面全体に対して割り当てられます。このとき、(*x1!*, *y1!*) は画面の左上角、(*x2!*, *y2!*) は画面の右下角のビューポート座標を示します。

参照　**CLS** ステートメント，**PMAP** 関数，**POINT** 関数，**SCREEN** ステートメント，**VIEW** ステートメント，**WIDTH** ステートメント

WindowState プロパティ

機能　実行時のフォームの表示状態を決めます。

構文　[*formname.*]**WindowState**[= *state%*]

state%　フォームの表示状態を示す値。0 が通常の大きさ，1 が最小化，2 が最大化を示します。デフォルトは 0 です。

対象　フォーム

解説　WindowState プロパティはデザイン時には無効で、必ず 0 に設定されます。実行時にコード中からフォームを最大化/最小化したいときに使います。フォームが最大化，最小化または復元されると、Resize イベントが発生します。また、フォームが最大化または復元されると、Paint イベントが発生します。

参照　MaxButton プロパティ，MinButton プロパティ，Paint イベント，Resize イベント

WRITE ステートメント

機能　数値または文字列データを画面に出力します。フォームが表示されている状態では利用できません。

構文　**WRITE** [, *expression*[, *expression*]...]

expression　出力するデータを示す数式または文字列式。省略すると、単に改行されます。複数の式を指定するときはカンマ (,) で区切ります。

解説　フォームが表示されているときに **WRITE** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

WRITE ステートメントは、**PRINT** ステートメントと同様に画面にデータを出力しますが、**PRINT** ステートメントと次のような点が異なります。

出力される各データ項目は、カンマで区切られます (カンマが実際に出力されます)。

文字列を出力するとき、ダブル クォーテーション(") で囲んで出力します。

数値を出力するときも、前後にスペースを空けません。

1 つの **WRITE #** ステートメントによる出力の最後では、必ず改行します。

TAB や **SPC** 関数を使うことができません。

参照　**PRINT** ステートメント，**WRITE #** ステートメント

WRITE # ステートメント

機能　数値または文字列データをシーケンシャル ファイルに書き込みます。

構文　**WRITE** *#filenumber%*[, *expression*[, *expression*]...]

filenumber%	オープンしているファイルのファイル番号。ファイルは、**OUTPUT** モードか **APPEND** モードでオープンされていなければなりません。
expression	出力するデータを示す数式または文字列式。省略すると、単に改行されます。複数の式を指定するときはカンマ (,) で区切ります。

解説　フォームが表示されているときに SCRN: または CONS: に対して **WRITE #** ステートメントを実行すると、"フォームが表示されているときは無効です" エラーが発生します。

WRITE # ステートメントは、**PRINT #** ステートメントと同様にファイルにデータを書き込みますが、**PRINT #** ステートメントと次のような点が異なります。

出力される各データ項目は、カンマで区切られます (カンマが実際に出力されます)。

文字列を出力するとき、ダブル クォーテーション(") で囲んで出力します。

数値を出力するときも、前後にスペースを空けません。

1 つの **WRITE #** ステートメントによる出力の最後では、必ず改行します。

TAB や **SPC** 関数を使うことができません。

LOCK ステートメントで書き込みを制限しているファイルにデータを書き込もうとすると、"書き込みできません" エラーが発生します。ただしこのエラーは、エラー処理ルーチンを使ってトラップできます。

参照　**PRINT #** ステートメント，**WRITE** ステートメント

YEAR 関数

機能　日付と時刻のシリアル値を評価して、1753～2078 までの範囲で、何年かを返します。

構文　**YEAR**(*serial*)

serial　日付と時刻を表すシリアル値 (-53,688～65,380)。

付録

A 予約語一覧

以下のリストは、Visual Basic for MS-DOS の予約語の一覧です。これらの単語そのものを、ラベル，変数やプロシージャなどの名前として使うことはできません。予約語には、関数，ステートメント，メソッド，演算子と、その中で使われるキーワードなど、Visual Basic for MS-DOS のインタープリタが認識するすべての単語が含まれています。アスタリスク (*) が付けてある単語は、QuickBasic バージョン 4.5 以降に新しく追加された予約語です。

ABS
ACCESS
ADDITEM*
ALIAS
ALL*
AND
ANY
APPEND
AS
ASC
ATN
BASE
BEEP
BEGINTRANS*
BINARY
BLOAD
BOF*
BSAVE
BYVAL
CALL
CALLS
CASE
CCUR*
CDBL
CDBL$
CDECL
CHAIN
CHDIR
CHDRIVE*
CHECKPOINT*
CHR$
CINT
CIRCLE
CLEAR
CLIPBOARD*
CLNG
CLOSE
CLS
COLOR
COM
COMMAND$
COMMITTRANS*
COMMON
CONST
CONTROL*
COS
CREATEINDEX*
CSNG
CSNG$
CSRLIN
CURDIR*
CURRENCY*
CVC*
CVD
CVDMBF
CVI
CVL
CVS
CVSMBF
DATA
DATE$
DATESERIAL*
DATEVALUE*
DAY*
DECLARE
DEF
DEFCUR*
DEFDBL
DEFINT
DEFLNG
DEFSNG
DEFSTR

DELETE*
DELETEINDEX*
DELETETABLE*
DIM
DIR$*
DO
DOEVENTS*
DOUBLE
DRAG*
DRAW
$DYNAMIC
ELSE
ELSEIF
END
ENDDOC*
ENDIF
ENVIRON
ENVIRON$
EOF
EQV
ERASE
ERDEV
ERDEV$
ERL
ERR
ERROR
ERROR$*
EVENT
EXIT
EXP
EXPLICIT*
FIELD
FILEATTR
FILES
FIX
FN
FOR
$FORM*
FORM*
FORMAT$*
FRE
FREEFILE
FUNCTION
GET
GETINDEX$*
GETTEXT*
GO
GOSUB
GOTO
HEX$
HIDE*
HOUR*
IF
IMP
$INCLUDE
INKEY$
INP
INPUT
INPUT$
INPUT¥
INPUTBOX$*
INSERT*
INSTR
INT
INTEGER
IOCTL
IOCTL$
IS
ISAM*
JIS$
KEXT$
KEY
KILL
KINSTR
KLEN
KMID$
KPOS
KTN$
LBOUND
LCASE$
LEFT$
LEN
LET
LINE
LIST
LOAD*
LOC
LOCAL
LOCATE
LOCK
LOF
LOG
LONG
LOOP
LPOS
LPRINT
LSET
LTRIM$
MID$
MINUTE*
MKC$*
MKD$
MKDIR
MKDMBF$
MKI$
MKL$
MKS$
MKSMBF$

MOD
MONTH*
MOVE*
MOVEFIRST*
MOVELAST*
MOVENEXT*
MOVEPREVIOUS*
MSGBOX*
NAME
NEWPAGE*
NEXT
NOT
NOW*
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
OUTPUT
PAINT
PALETTE
PCOPY
PEEK
PLAY
PMAP
POINT
POKE
POS
PRESERVE*
PRESET
PRINT
PRINTER*
PRINTFORM*
PSET
PUT
QBCOLOR*
RANDOM
RANDOMIZE
READ
REDIM
REFRESH*
REM
REMOVEITEM*
RESET
RESTORE
RESUME
RETRIEVE*
RETURN
RGB*
RIGHT$
RMDIR
RND
ROLLBACK*
RSET
RTRIM$
RUN
SADD
SAVEPOINT*
SCREEN
SECOND*
SEEK
SEEKEQ*
SEEKGE*
SEEKGT*
SEG
SELECT
SETFOCUS*
SETINDEX*
SETMEM
SETTEXT*
SGN
SHARED
SHELL*
SHOW*
SIGNAL*
SIN
SINGLE
SLEEP
SOUND
SPACE$
SPC
SQR
SSEG*
SSEGADD*
STACK*
$STATIC
STATIC
STEP
STOP
STR$
STRING
STRING$
SUB
SWAP
SYSTEM
TAB
TAN
TEXTCOMP*
TEXTHEIGHT*
TEXTWIDTH*
THEN
TIME$
TIMER
TIMESERIAL*
TIMEVALUE*
TO

TROFF
TRON
TYPE
TYPEOF*
UBOUND
UCASE$
UEVENT
UNLOAD*
UNLOCK
UNTIL
UPDATE*
USING
VAL
VARPTR
VARPTR$
VARSEG
VIEW
WAIT
WEEKDAY*
WEND
WHILE
WIDTH
WINDOW
WRITE
XOR
YEAR*

B データ型

Visual Basic がサポートしている基本データ型には、次のようなものがあります。

型宣言文字	データ型	サイズ	範囲
%	整数 **INTEGER**	2 バイト	-32,768～32,767
&	長整数 **LONG**	4 バイト	-2,147,483,648～ 2,147,483,647
!	単精度浮動小数点数 **SINGLE**	4 バイト	-3.402823E+38～ -1.401298E-45, 0, 1.401298E-45～3.402823E+38
#	倍精度浮動小数点数 **DOUBLE**	8 バイト	-1.797693134862315D+308～ -4.94066D-324, 0, 4.94066D-324～ 1.797693134862315D+308
@	通貨型数 **CURRENCY**	8 バイト	-922337203685477.5808～ 922337203685477.5807
$	文字列 **STRING**	最大 32767 バイト	
なし	ユーザー定義型	各要素のサイズの合計	
なし	フォーム **FORM**	"	
なし	コントロール **CONTROL**	"	

INTEGER (整数型)

整数は、16 ビット (2 バイト) の 2 の補数として表されます。扱える値の範囲は -32,768～32,767 です。

整数は、この範囲を表す以外に、ブール値 (2 つの状態だけを示す値) を表すのにも使われます。ブール値では、-1 (または 0 以外) が真を表し、0 が偽を表します。また、列挙型の値を示すためにも使うことができます。列挙型は、重複のない整数の、ある決められた組み合わせを含むことができ、そのそれぞれが、使われている特性に関する特別な意味を持ちます。列挙型の値は、すでに決まっている選択肢の中から選んで行くときに便利です。

重　要　いくつかのプロパティは、オン/オフを示す2つの値だけを取ります。デザイン時にこれらのプロパティを設定するとき、フォーム デザイナのプロパティ バーには True または False と表示されます。コード中では、True (真) を -1、False (偽) を 0 として扱ってください。コード中でこのような値 (ブール値) を扱うとき、記号定数として TRUE および FALSE を定義しておくと便利です。Visual Basic があらかじめ用意しているインクルード ファイル CONSTANT.BI には、このようにコード中で便利な記号定数が定義されています。$INCLUDE メタコマンドを使ってプログラムにインクルードしてください。

LONG (長整数型)

長整数は 32 ビット (4 バイト) の 2 の補数として表されます。扱える値の範囲は -2,147,483,648 ～ 2,147,483,647 です。

SINGLE (単精度浮動小数点数型)

単精度浮動小数点数は 32 ビット (4 バイト) で表されます。扱える値の範囲は -3.402823E+38 ～ -1.401298E-45, 0, 1.401298E-45 ～ 3.402823E+38 です。

浮動小数点数は、内部的に IEEE 形式の数値として扱われ、符号，指数部，仮数部の 3 つの部分で構成されています。単精度浮動小数点数は、符号 1 ビット，指数部 8 ビット，仮数部 23 ビットと、暗黙の 1 ビットで表されます。

DOUBLE (倍精度浮動小数点型)

倍精度浮動小数点数は 64 ビット (8 バイト) で表されます。扱える値の範囲は -1.797693134862315D+308 ～ -4.94066D-324, 0, 4.94066D-324 ～ 1.797693134862315D+308 です。

浮動小数点数は、内部的に IEEE 形式の数値として扱われ、符号，指数部，仮数部の 3 つの部分で構成されています。単精度浮動小数点数は、符号 1 ビット，指数部 11 ビット，仮数部 52 ビットと、暗黙の 1 ビットで表されます。

CURRENCY (通貨型)

通貨型は、10,000 倍されて 64 ビット (8 バイト) の 2 の補数表現の整数として記憶される、15 桁の整数部分と 4 桁の小数部分を持つ固定小数点数です。扱える値の範囲は -922,337,203,685,477.5808 ～ 922,337,203,685,477.5807 です。

通貨型は、外国通貨の計算や、正確さが特に求められる固定小数点数の計算に便利です。

STRING (文字列型)

文字列の基本データ型には、あらかじめ記憶領域のサイズが決まっている固定長文字列と、文字列の長さによって使用するメモリ サイズが変化する可変長文字列があります。文字列定数は、常に固定長として扱われます.

- 可変長文字列

  可変長文字列の最大長は、32,767 バイトです。ただし文字列の格納に若干のオーバーヘッドが必要なため、実際に格納できるのは 32K バイト以下です。

- 固定長文字列

  固定長文字列は、宣言されたバイト数だけの長さを持ちます。最大 32,767 バイトです。

CONTROL (コントロール型)

コントロール型は、Visual Basic のプロシージャのパラメータ (仮引数) としてだけ宣言できます。プロシージャの宣言の中で、**CONTROL** は次のように使われます。

```
SUB ChangeText (AnyControl AS CONTROL, NewValue$)
    IF TYPE OF AnyControl IS TextBox THEN
        AnyControl.Text = NewValue$
    END IF
END SUB
```

FORM (フォーム型)

フォーム型は、Visual Basic のプロシージャのパラメータ (仮引数) としてだけ宣言できます。プロシージャの宣言の中で、**FORM** は次のように使われます。

```
SUB ChangeCaption (EmployeeInfo AS FORM, NewCaption$)
    EmployeeInfo.Caption = NewCaption$
END SUB
```

C 演算子

次の表は、Visual Basic の算術演算子, 関係演算子, 論理演算子を優先順位の高い方から並べたものです。

タイプ	機能	シンボル
算術演算子	指数演算	^
	マイナス符号	-
	乗算と除算	*, /
	整数除算	¥
	剰余演算	MOD
	加算と減算	+, -
関係演算子	等しい, より大きい, より小さい, 等しくない, 以上, 以下	=, >, <, <>, >=, <=
論理演算子 (ビットごと)	論理否定	NOT
	論理積	AND
	論理和	OR
	排他的論理和	XOR
	同値	EQV
	包含	IMP

このあと、論理演算子について、真理値表と合わせて順に解説して行きます。論理演算子は、次のような構文で使います。

結果 = 数式1 論理演算子 数式2

AND 演算子

AND 演算子は、2 つ以上の関係式の評価に使う、論理積演算子です。両方の式の値が真 (-1) の場合だけ結果が真になり、片方または両方とも真でなければ、結果は偽 (0) になります。

AND 演算子は、ビットごとの比較を行う演算子としても使うことができます。この場合、AND 演算子は 2 つの数値の対応するビットを比較し、両方とも 1 のビットだけを 1 にします。

数式 1 のビット	数式 2 のビット	結果のビット
0	0	0
0	1	0
1	0	0
1	1	1

EQV 演算子

EQV 演算子は 2 つの数値の対応するビットを比較し、両方のビットが同じ場合だけ 1 にします。

数式 1 のビット	数式 2 のビット	結果のビット
0	0	1
0	1	0
1	0	0
1	1	1

IMP 演算子

IMP 演算子は 2 つの数値の対応するビットを比較し、次のような結果を返します。

数式 1 のビット	数式 2 のビット	結果のビット
0	0	1
0	1	1
1	0	0
1	1	1

NOT 演算子

NOT 演算子は与えられた数値の各ビットを反転します。

数式のビット	結果のビット
0	1
1	0

NOT 演算子は、数値変数のビットの値を反転させるため、もし変数の値が偽 (0) なら真 (-1), 真なら偽の結果を得ることができます。

OR 演算子

OR 演算子は、2 つ以上の関係式の評価に使う、論理和演算子です。片方でも式の値が真 (-1) なら結果は真になり、両方とも偽 (0) のときだけ結果が偽になります。

OR 演算子は、ビットごとの比較を行う演算子としても使うことができます。この場合、OR 演算子は 2 つの数値の対応するビットを比較し、片方でもビットが 1 なら 1 にします。

数式 1 のビット	数式 2 のビット	結果のビット
0	0	0
0	1	1
1	0	1
1	1	1

XOR 演算子

XOR 演算子は、2 つ以上の関係式の評価に使う、論理排他的論理和演算子です。片方の式の値が真 (-1) でもう一方が偽 (0) なら結果は真になり、両方とも真または偽なら結果は偽になります。

XOR 演算子は、ビットごとの比較を行う演算子としても使うことができます。この場合、XOR 演算子は 2 つの数値の対応するビットを比較し、片方のビットが 1 の場合だけ 1 にします。

数式 1 のビット	数式 2 のビット	結果のビット
0	0	0
0	1	1
1	0	1
1	1	0

D エラー メッセージ一覧

ここでは、次のエラー メッセージについて説明します。

- プログラム中からトラップ可能な実行時エラー
- Basic コンパイラ (Microsoft Basic Compiler, BC.EXE) のエラー
- LIB (Microsoft Library Manager, LIB.EXE) のエラー
- LINK (Microsoft Segmented-Executable Linker, LINK.EXE) のエラー

D.1 トラップ可能な実行時エラー

表 D.1 は、実行時にユーザー プログラムからトラップ可能なエラーの一覧です。発生したエラーのエラー コードは、ERR 関数によって得ることができます。プログラム中でエラーを処理するときは、エラー メッセージのテキストではなく、このエラー コードを元にエラーの種類を判断してください。

表 D.1 トラップ可能なエラー

エラー コード	エラー メッセージ
1	FOR がありません
2	構文エラーです
3	GOSUB がありません
4	DATA がありません
5	引数が有効範囲にありません
6	オーバーフローしました
7	メモリが不足です
8	ラベルが未定義です
9	添字が有効範囲にありません
10	宣言が重複しています
11	0 で除算しました
12	イミディエイト ウィンドウでは使えません
13	型が一致しません
14	文字列領域が不足です
16	文字式が複雑すぎます
17	継続できません

表 D.1　トラップ可能なエラー (続き)

エラー コード	エラー メッセージ
18	関数が未定義です
19	RESUME がありません
20	エラー未発生で RESUME は実行できません
24	デバイス タイムアウトです
25	デバイスが異常です
26	NEXT がありません
27	用紙がありません
29	WEND がありません
30	WHILE がありません
33	ラベルが重複しています
35	定義されていません
37	引数の個数が一致しません
38	配列が定義されていません
40	変数が必要です
50	FIELD 長を超えました
51	内部エラーです
52	ファイルの名前または番号が不正です
53	ファイルが見つかりません
54	ファイル モードが不正です
55	すでにオープンされています
56	FIELD で指定した変数は使用できません
57	デバイス I/O エラーです
58	すでに同名のファイルが存在しています
59	レコード長が一致しません
61	ディスクの空き容量が不足です
62	ファイルにもうデータがありません
63	レコード番号が無効です
64	ファイル名が無効です
67	ファイル数が多すぎます
68	デバイスが準備されていません
69	通信バッファがオーバーフローしました
70	書き込みできません
71	ディスクが準備されていません
72	ディスク エラーです
73	この機能はサポートされていません
74	ディスク間のリネームはできません
75	パス名が無効です
76	パスが見つかりません

表 D.1 トラップ可能なエラー (続き)

エラー コード	エラー メッセージ
80	スタブで削除されています
81	ISAM - 無効な名前です
82	ISAM - テーブルが見つかりません
83	ISAM - インデックスが見つかりません
84	ISAM - 列が不正です
85	ISAM - レコードがありません
86	ISAM - 重複のないインデックスで値が重複しています
87	ISAM - NULL インデックスにこの処理はできません
88	ISAM - このデータベースは修復が必要です
89	ISAM - バッファが不足です
260	これ以上タイマーを有効にできません
271	スクリーン モードが不正です
272	フォームが表示されているときは無効です
340	コントロール配列の要素が存在しません
341	オブジェクト配列のインデックス番号が不正です
342	コントロール配列のためのメモリが不足です
343	オブジェクトが配列ではありません
344	オブジェクト配列にはインデックス番号が必要です
345	このフォームにこれ以上のコントロールを作成できません
360	このオブジェクトはすでにロードされています
361	このオブジェクトはロードもアンロードもできません
362	フォーム デザイナで作成したコントロールはアンロードできません
364	すでにアンロードされています
365	現在アンロードできません
380	プロパティの値が不正です
381	プロパティ配列のインデックス番号が不正です
382	実行時にはこのプロパティを設定できません
383	読み込み専用のプロパティです
384	フォーム最大, 最小時はプロパティ値を変更できません
385	プロパティ配列にはインデックス番号が必要です
386	このプロパティは実行時には無効です
387	このコントロールには使用できないプロパティです
400	表示されているフォームをモーダルにすることはできません
401	モーダルなフォームが表示されているときは表示できません
402	最も上のモーダルなフォームをクローズしてください
403	MDI フォームはモーダルにできません
410	MDI フォームの WindowState プロパティは変更できません

表 D.1　トラップ可能なエラー (続き)

エラー コード	エラー メッセージ
420	オブジェクトがロードされていません
421	このオブジェクトには使用できないメソッドです
422	このコントロールには使用できないプロパティです
423	このフォームには使用できないプロパティかコントロールです
424	オブジェクトの指定が必要です
425	オブジェクトの使い方が不正です
430	アクティブなコントロールがありません
431	アクティブなフォームがありません
480	メモリ不足のため AutoRedraw は使えません

D.2　BC コンパイラのエラー

0 で除算しました (ERR: 11)

数式内で、0 で除算しています。

解決するには次のようにしてください。

- 変数の値や計算の途中の結果が 0 になっていないかどうか確認する。
- エラーに対するチェック ルーチンを組み込む。

1 ～ 32767 の整数を指定してください

このステートメントには、1～32,767 の範囲の整数の引数が必要です。

ALIAS には文字列定数の引数が必要です

DECLARE ステートメントの ALIAS キーワードには、文字列定数の引数が必要です。

AS 節がありません

AS 節を使って宣言された変数が、AS 節なしで参照されています。

その変数の最初の宣言で AS 節を使っていれば、あとの DIM, REDIM, SHARED, COMMON ステートメントでも AS 節を付けなければなりません。

最初の宣言の AS 節を削除するか、あとの宣言でも AS 節を付けるようにしてください。

AS がありません

次のようなステートメントでは、AS キーワードが必要です。

```
OPEN "FILENAME" FOR INPUT AS #1
```

BASE がありません

OPTION BASE ステートメントには、BASE キーワードが必要です。

CALLS では SEG, BYVAL は使えません

SEG や BYVAL は、CALL ステートメントの中でのみ使うことができます。CALLS ステートメントでは使えません。

COMMON のブロック名が不適切です

名前付きの COMMON ブロック中に、不正なブロック名 (例えば Basic の予約語) が指定されています。

COMMON ブロックが多すぎます

1 つのプログラムで使うことができる名前付き COMMON ブロックは 126 までです。

CONST, DIM SHARED は SUB, FUNCTION の前に置いてください (警告)

CONST, DIM SHARED ステートメントは、SUB や FUNCTION プロシージャを定義する前に置く必要があります。

BC コンパイラでは、このエラーは致命的ではありません。つまりプログラムを実行することはできますが、結果は保証されません。

DECLARE が必要です

暗黙の SUB, FUNCTION プロシージャ呼び出し (CALL ステートメントを使わないプロシージャ呼び出し) が、プロシージャを定義する前に行われています。あらかじめプロシージャを定義または宣言しておいてください。

DEF FN またはプロシージャのエラーです

このエラーは次のような状況で発生します。

- SUB, FUNCTION プロシージャ、DEF FN ブロック内で定義したラベルに、そのプロシージャまたはブロックの外から分岐しようとした。
- SUB, FUNCTION プロシージャ内で、行番号または行ラベルを指定した RETURN ステートメントを実行しようとした。
- SUB, FUNCTION プロシージャ、DEF FN ブロックの外で、STATIC ステートメントを実行しようとした。または、SUB, FUNCTION プロシージャの外で SHARED ステートメントを実行しようとした。
- SUB, FUNCTION プロシージャ、DEF FN ブロック内で、ファイル名を指定しない RUN ステートメントを実行しようとした。

DEF がありません

END DEF に対応する DEF がありません。DEF FN ステートメントを挿入するか、END DEF を削除します。

DEFxxx の型指定が不正です

DEF*type* ステートメントの指定に誤りがあります。DEF*type* ステートメントの範囲指定は、アルファベットを使わなければなりません。

DO がありません

LOOP に対応する DO がありません。

解決するには次のようにしてください。

- DO...LOOP 構造の先頭に DO を挿入する。
- DO...LOOP ブロック中の他の制御構造が正しく対になっているかどうか確認する。

DO...LOOP の内側になければなりません

DO...LOOP 構造の外側で、EXIT DO が使われています。EXIT DO を削除するか、DO...LOOP 構造の中に移してください。

END DEF がありません

複数行にわたる関数定義で、対応する END DEF がありません。DEF FN ブロックの最後に、END DEF ステートメントを加えてください。

END IF がありません

IF に対応する END IF がありません。

解決するには次のようにしてください。

- IF...END IF 構造の最後に END IF を挿入する。
- IF...END IF ブロック中の他の制御構造が正しく対になっているかどうか確認する。

END SELECT がありません

SELECT CASE に対応する END SELECT がありません。

解決するには次のようにしてください。

- SELECT...END SELECT 構造の最後に END SELECT を挿入する。
- SELECT...END SELECT ブロック中の他の制御構造が正しく対になっているかどうか確認する。

END SUB, END FUNCTION がありません

SUB に対応する END SUB、FUNCTION に対応する END FUNCTION がありません。プロシージャの最後に END SUB または END FUNCTION を挿入してください。

EVENT ON を含むプログラムは /V または /W を指定してコンパイルしてください (警告)

EVENT ON ステートメントを含むプログラムをコンパイルするには、コマンドライン上で /V または /W オプションを指定する必要があります。これは、単なる警告メッセージです。

- ON KEY や ON COM などの ON *event* ステートメントを含むプログラムでは、正しく実行するためには、BC コマンドライン上で /V または /W オプションを必ず指定しなければなりません。
- ON *event* ステートメントを含まないプログラムでは、EVENT ON を EVENT OFF に直し、/V または /W オプションは指定しないでください。

FOR がありません (ERR: 1)

NEXT に対応する FOR がありません。

解決するには次のようにしてください。

- FOR...NEXT 構造の先頭に FOR を挿入する。
- FOR...NEXT ブロック中の他の制御構造が正しく対になっているかどうか確認する。

FOR...NEXT の内側になければなりません

FOR...NEXT 構造の外側で、EXIT FOR が使われています。EXIT FOR を削除するか、FOR...NEXT 構造の中に移してください。

GOSUB がありません

ON *event* ステートメント中に GOSUB がありません。

GOTO がありません

ON ERROR ステートメント中に GOTO がありません。

GOTO または GOSUB が必要です

GOTO または GOSUB ステートメントが必要です。

IF がありません

ELSE に対応する IF がありません。このエラーは、IF ステートメントのネストに誤りがあるときにも起こります。

解決するには次のようにしてください。

- IF...END IF 構造の先頭に IF を挿入する。
- IF...END IF ブロック中の他の制御構造が正しく対になっているかどうか確認する。

IF がありません

ELSEIF に対応する IF がありません。このエラーは、IF ステートメントのネストに誤りがあるときにも起こります。

IF がありません

END IF に対応する IF がありません。

解決するには次のようにしてください。

- IF...END IF 構造の先頭に IF を挿入する。
- IF...END IF ブロック中の他の制御構造が正しく対になっているかどうか確認する。

IF...THEN...ELSE 内にある制御構造が不完全です

1 行形式の IF...THEN...ELSE ステートメント中に、NEXT, WEND, END IF, END SELECT, LOOP などがあります。

INPUT がありません

INPUT キーワードが必要です。

INPUT の変数が多すぎます

INPUT ステートメントで使うことができる変数は 60 個までです。

IS がありません

SELECT CASE ステートメントの CASE 節に、IS キーワードがありません。

ISAM - 無効な型です

ISAM の OPEN ステートメントで、テーブルのタイプを示すユーザー定義型は、次の規則に従って指定しなければなりません。

- フィールド名は 30 バイト以内
- SINGLE データ型の要素を含まない

Input file not found

BC コマンドラインで指定したソース ファイルが見つからないので、コンパイルを中止しました。

ソース ファイルのスペルと、ソース ファイルが置かれているディレクトリを確認し、入力し直してください。

LINE INPUT の変数は複数指定できません

LINE INPUT ステートメントで指定できる変数は 1 つだけです。

LOOP がありません

DO に対応する LOOP がありません。

解決するには次のようにしてください。

- DO...LOOP 構造の最後に LOOP を挿入する。
- DO...LOOP ブロック中の他の制御構造が正しく対になっているかどうか確認する。

NEXT がありません (ERR: 26)

FOR に対応する NEXT がありません。

ON ERROR を含むプログラムは /E を指定してコンパイルしてください

ON [LOCAL] ERROR ステートメントを含むプログラムをコンパイルするには、コマンドライン上で /E オプションを指定する必要があります。

ON event を含むプログラムは /V または /W を指定してコンパイルしてください

ON KEY, ON TIMER, ON COM などの ON *event* ステートメントを含むプログラムをコンパイルするには、コマンドライン上で /V または /W オプションを指定する必要があります。/V オプションを指定すると、各ステートメントごとにイベントの発生をチェックし、/W オプションを指定すると、ラベルの付いた行ごとにチェックします。

RESUME を含むプログラムには /X を指定してコンパイルしてください

RESUME, RESUME NEXT, RESUME 0 ステートメントを含むプログラムをコンパイルするには、コマンドライン上で /X オプションを指定する必要があります。

SELECT CASE と CASE の間にステートメントやラベルは許されません

SELECT CASE と最初の CASE の間には、ステートメントや行ラベルを置くことはできません。ただし、コメントとステートメントの区切り文字は置くことができます。

SELECT がありません

END SELECT に対応する SELECT CASE がありません。

解決するには次のようにしてください。

- SELECT...END SELECT 構造の先頭に SELECT CASE を挿入する。
- SELECT...END SELECT ブロック中の他の制御構造が正しく対になっているかどうか確認する。

SUB, FUNCTION がありません

END SUB, END FUNCTION に対応する SUB, FUNCTION がありません。編集中に間違って削除していないか確認してください。

SUB, FUNCTION がありません

DECLARE ステートメントに対応するプロシージャがありません。

SUB, FUNCTION の外では使えません

このステートメントは、モジュール レベルおよび DEF FN ブロック中では使えません。強調表示されているステートメントを、SUB または FUNCTION プロシージャ中に移してください。フォーム モジュールのモジュールレベルには実行可能ステートメントを置けないことに注意してください。

SUB, FUNCTION 内のローカル文字列が多すぎます

プロシージャ内のローカル文字列の数が 255 を超えています。

SUB, FUNCTION, DEF FN の外では使えません

このステートメントは、モジュール レベルでは使えません。強調表示されているステートメントを、SUB または FUNCTION プロシージャ，DEF FN ブロック中に移してください。フォーム モジュールのモジュールレベルには実行可能ステートメントを置けないことに注意してください。

SUB, FUNCTION, DEF FN の中では使えません

このステートメントは、SUB または FUNCTION プロシージャ，DEF FN ブロックの中では使えません。強調表示されているステートメントをモジュール レベルに移してください。フォーム モジュールのモジュールレベルには実行可能ステートメントを置けないことに注意してください。

THEN がありません

THEN キーワードが必要です。

TO がありません

TO キーワードが必要です。

TYPE がありません

END TYPE に対応する TYPE がありません。

WEND がありません (ERR: 29)

WHILE に対応する WEND がありません。

解決するには次のようにしてください。

- WHILE...WEND 構造の最後に WEND を挿入する。
- WHILE...WEND ブロック中の他の制御構造が正しく対になっているかどうか確認する。

WHILE がありません (ERR: 30)

WEND に対応する WHILE がありません。

解決するには次のようにしてください。

- WHILE...WEND 構造の先頭に WHILE を挿入する。
- WHILE...WEND ブロック中の他の制御構造が正しく対になっているかどうか確認する。

DO...LOOP ステートメントを使うと、WHILE...WEND ステートメントより柔軟にループを操作できます。

アスタリスクで長さ指定してください

ユーザー定義型の文字列の定義にアスタリスクがありません。

インクルード ファイルのアクセス エラーです

$INCLUDE メタコマンドで指定されたインクルード ファイルが見つかりません。

オーバーフローしました

計算結果が大きすぎて、Basic の数値形式で表すことができません。

型が一致しません (ERR: 13)

このエラーは次のような状況で発生します。

- 指定された変数が求められているデータ型でない。例えば、文字列変数と数値変数に対して SWAP ステートメントを実行しようとした。または 2 つの数値のデータ型が異なる。
- RUN ステートメントで英数字の行ラベルを指定しようとした。
- フォーム コード中で、指定されたオブジェクトが見つからない。またはそのオブジェクトに対して明示的に参照する必要がある。

- 複数のモジュールで構成されるプログラムで、COMMON ステートメントが適切に対応していない。
- ISAM の OPEN FOR ISAM ステートメントで、ユーザー定義型の要素のデータ型がテーブルの列のデータ型と一致しない。
- ISAM の INSERT ステートメントで、ユーザー定義型の要素数がテーブルの列数と一致しない。または、ユーザー定義型の要素のデータ型がテーブルの列のデータ型と一致しない。
- ISAM の RETRIEVE ステートメントで、ユーザー定義型の要素のデータ型がテーブルの列のデータ型と一致しない。
- ISAM の SEEK*operand* ステートメントで、*keyvalue* 引数のデータ型がカレント インデックスのデータ型と一致しない。
- ISAM の UPDATE ステートメントで、更新しようとしているテーブルと、指定したユーザー定義型が一致しない。

可変長文字列を指定してください

プロシージャの仮引数や FIELD ステートメントでは、固定長文字列は使えません。

仮引数が重複しています

SUB または FUNCTION プロシージャの宣言で、仮引数が重複しています。

```
SUB GetName(A,B,C,A) STATIC
```

仮引数が不正です

SUB, FUNCTION のパラメータリスト内にエラーがあります。

関数が未定義です (ERR: 18)

関数が参照されましたが、定義されていません。

解決するには次のようにしてください。

- 呼び出そうとした関数の名前のスペルを確認する。
- DECLARE ステートメントで FUNCTION プロシージャを宣言し、引数のデータ型チェックが行われるようにする。

関数の引数が多すぎます

SUB, FUNCTION プロシージャや DEF FN 関数に引き渡すことができる引数は 60 個までです。

関数定義がネストしています

SUB, FUNCTION プロシージャ、DEF FN 関数が、他の SUB, FUNCTION プロシージャ、DEF FN 関数の中にあります。

関数名が不正です

DEF FN 関数の名前は、FN で始める必要があります。

カンマがありません

カンマが必要です。

行が長すぎます

1 行の文字数は 255 文字までです。

行頭のみ許されるステートメントです

IF...END IF 構造で、IF, ELSE, ELSEIF, END IF の前に置けるのは、行番号または行ラベルだけです。また、SELECT...END SELECT 構造で、CASE, END SELECT の前に置けるのも、行番号または行ラベルだけです。SELECT や IF の前にあるステートメントを削除してください。

行番号またはラベルがありません

行ラベルまたは行番号が必要なステートメント (GOTO など) で、それらが指定されていません。

区切り文字が不正です

PRINT USING, WRITE ステートメントの中に無効な区切り文字があります。区切り文字には、セミコロンまたはカンマを使ってください。

構文エラーです (ERR: 2)

このエラーは次のような状況で発生します。

- コンパイル時に、Basic のキーワードや引数の入力に誤りがあった。
- 実行時に、DATA ステートメントの形式に誤りがあった。
- フォーム コード中で、指定されたオブジェクトが見つからない。またはそのオブジェクトに対して明示的に参照する必要がある。
- ISAM を使うプログラムで、SEEKGT, SEEKGE, SEEKEQ ステートメントのキー値がカレント インデックスのキー値と異なっている。

固定長文字列は使えません

固定長文字列を SUB または FUNCTION プロシージャの仮引数として使おうとしています。SUB または FUNCTION プロシージャの呼び出しと DECLARE ステートメントを比較して、矛盾がないかどうか確認してください。

このオブジェクトには使用できないメソッドです (ERR: 421)

指定されたオブジェクトに対して適用できないメソッドを使おうとしました。例えば、ADDITEM メソッドが適用できるオブジェクトはリスト ボックスとコンボ ボックスだけであるため、Command1.ADDITEM と指定するとエラーになります。

解決するには次のようにしてください。

- オンライン ヘルプのメソッドのトピックから、適用できるオブジェクトを調べ、必要な変更を加える。
- オンライン ヘルプのオブジェクトのトピックから、そのオブジェクトに適用できるメソッドを調べ、必要な変更を加える。

このオブジェクトには適用できないプロパティです

指定されたオブジェクトには、そのプロパティは存在しません。オンライン ヘルプまたはリファレンス マニュアルで、適切なプロパティを確認してください。

この機能はサポートされていません (ERR: 73)

サポートされていない機能を使おうとしました。例えば、適切な ISAM の TSR (PROISAM または PROISAMD) をロードせずに ISAM の 機能を使おうとしたり、EMS メモリが利用できないときに FRE(-3) を実行しようとすると、このエラーが発生します。

また、このバージョンの Visual Basic ではサポートされていない、他のバージョンの Basic の機能を使おうとしたときや、現在使用している OS では使うことができない機能を使おうとしたときも、このエラーが起こります。

この識別子にピリオドは使えません

ユーザー定義型の識別子や要素の名前にピリオドを含めることはできません。また、名前のピリオドより前の部分が、プログラム中の AS *usertype* 節の中で使われているときは、変数名にピリオドを使うことはできません。

原則として変数名にピリオドを含めることは可能ですが、ユーザー定義型の要素を指定するときのセパレータ以外には使わないことをおすすめします。既存のプログラムで変数名中にピリオドを使っているときは、大文字と小文字の組み合せに変更するようにしてください。例えば、変数 ALPHA.BETA は AlphaBeta にします。

最初の宣言で AS 節が必要です

AS 節を使わずに宣言された変数が、AS 節を使って参照されています。

最初の宣言に AS 節を付けるか、あとの宣言の AS 節を削除してください。

式が複雑すぎます

何らかの Visual Basic の内部的な限界を超えた処理が行われました。例えば、数式を評価するとき、変数に代入されていない文字列は一時的な領域に格納されます。このような文字列が多いと、このエラーが起こることがあります。また、数式が複雑すぎても、このエラーが起こります。式を単純化するか、文字列を変数に代入してください。

識別子を指定してください

識別子が必要な場所に、数値または Basic の予約語を使おうとしています。

次元が多すぎます

配列で指定できる次元は 60 までです。

実行文の前に置いてください

$FORM メタコマンド, COMMON, DECLARE, OPTION EXPLICIT ステートメントの位置が間違っています。$FORM, COMMON, DECLARE, OPTION EXPLICIT は、実行可能ステートメントより前に置かなければなりません。

次のものを除くすべてのステートメントは実行可能ステートメントです。

- COMMON
- CONST
- DATA
- DECLARE
- DEF*type*
- DIM (静的配列に対する場合)
- OPTION BASE
- OPTION EXPLICIT
- REM
- SHARED
- STATIC
- TYPE
- すべてのメタコマンド

数値定数が大きすぎます

数値定数の値が、そのデータ型で扱える範囲を超えています。

数値定数に無効な文字が使われています

数値定数に不適当な型宣言文字が含まれています。

数値定数の形式に誤りがあります

数値定数が、適切な形式になっていません。

ステートメントを無視しました (警告)

TRON および TROFF ステートメントを含むプログラムを、/D オプションを指定せずにコンパイルしようとしました。BC コンパイラでは、このエラーは致命的ではありません。つまりプログラムを実行することはできますが、結果は保証されません。

すでに FOR のループ カウンタとして使われています

ネストした FOR...NEXT 構造の中で、同じループ カウンタを重複して使っています。各 FOR...NEXT 構造に対するループ カウンタに、別の変数を使うようにしてください。

制御ブロック中でプロシージャや DEF FN 関数は定義できません

SUB, FUNCTION プロシージャや DEF FN ブロックは、IF...END IF, SELECT...END SELECT などの制御構造の中では定義できません。

セミコロンがありません

セミコロン (;) が必要です。

宣言が重複しています (ERR: 10)

このエラーは次のような状況で発生します。

- プログラムの実行中に静的配列の次元を変えようとした。
- ISAM を使うプログラムで、すでに存在するインデックスを生成しようとした。

解決するには次のようにしてください。

- プログラム実行中に配列の次元を変更しようとする場合は、動的配列を使い、REDIM ステートメントで変更するようにする。
- データを保持したまま配列のサイズを変えるには、REDIM PRESERVE ステートメントを使う。

代替数値演算ライブラリでは通貨型は使用できません

代替数値演算ライブラリを使っているときに通貨型 (CURRENCY) を使おうとしました。/FPa オプションを指定してコンパイルするときは、プログラム中に次のものが含まれていないようにしなければなりません。

- アットマーク (@) 型宣言文字
- DEFCUR ステートメント
- CURRENCY キーワード

データ メモリがオーバーフローしました

プログラム データが多すぎてメモリに格納できません。このエラーは、定数が多すぎたり、静的配列データが大きすぎるときによく起こります。コンパイル時にこのエラーが起こったときは、デバッグ オプションをオフにしてみてください。それでもメモリが足りないときは、プログラムを複数のモジュールに分割するか、/AH と /FS オプションを使ってください。

等号がありません

等号 (=) が必要です。

動的配列の要素は指定できません

VARPTR$ 関数には、動的配列の要素を指定できません。

認識できないステートメントです

おそらく Basic ステートメントのタイプ ミスと思われます。

配列が大きすぎます

配列の宣言のためのユーザー データ領域が不足しています。配列のサイズを減らすか、プログラムの先頭で $DYNAMIC メタコマンドを使い、動的配列を使うようにしてください。

このエラーはまた、/AH オプションを指定せずにコンパイルしたプログラムで、サイズが 64K を超える静的配列を利用したときにも発生します。この場合、配列のサイズを減らすか、動的配列を使うか、/AH オプションを指定してコンパイルするようにしてください。

配列が定義されていません (ERR: 38)

現在定義されていない配列を使おうとしました。

配列が DIM, REDIM ステートメントで定義されているかどうか、確認してください。

配列の下限が上限を超えています

DIM ステートメントで、配列の添字の最小値に、最大値よりも大きな値を指定しています。

配列の次元が一致していません

配列の参照で次元に誤りがあります。配列の宣言をチェックし、参照の方法を変更してください。

配列の次元が宣言されていません (警告)

宣言されていない配列を参照しようとしました。

BC コンパイラでは、このエラーは致命的ではありません。つまりプログラムを実行することはできますが、結果は保証されません。

配列の添字に構文の誤りがあります

配列の添字に構文エラーがあります。例えば、配列の添字に文字列型と整数型の両方のデータ型が含まれていると、このエラーが起こります。

解決するには次のようにしてください。

- Basic のキーワードや引数の指定にタイプ ミスや形式の間違いがないかどうか確認する。
- フォーム コードのオブジェクトを明示的に参照する。
- SEEKxx ステートメントのキー値とカレント インデックスの値の数を調べ、必要があれば再コンパイルする。

配列はすでに宣言されています

このエラーは次のような状況で発生します。

- 同じ静的配列を宣言する複数の DIM ステートメントがある。
- 配列の宣言のあとに、OPTION BASE ステートメントが置かれている。
- 静的変数に対して REDIM ステートメントを使おうとした。
- 宣言せずに配列を使った後で、DIM ステートメントで宣言しようとした。

バイナリ ソースのためのリスティングが作成できません (警告)

バイナリ形式のソース ファイルを、/A オプションを指定してコンパイルしようとしました。/A オプションなしで再コンパイルしてください。

バイナリ形式ソース ファイル (警告)

この警告は次のような状況で表示されます。

- コンパイルしようとしたファイルが、このバージョンの Basic と互換性がないとき。バイナリ ファイルを ASCII ファイルにセーブし直して、再度コンパイルしてください。
- バイナリ形式のソース ファイルをコンパイルするときに、/ZI, /ZD CodeView オプションを指定したとき (この場合は警告メッセージが表示されるだけです)。

パス名が無効です (警告)

このエラーは次のような状況で発生します。

- OPEN, MKDIR, CHDIR, RMDIR ステートメントを実行するとき、指定されたパスが見つからなかった。
- すでに存在する読み込み専用ファイルに上書きしてセーブしようとした。プロジェクト (.MAK) ファイルが読み込み専用なら、プログラム全体がセーブできない。

解決するには次のようにしてください。

- プログラミング環境で、テキスト ボックスに入力したファイル名が正しく書式づけられているか確認する。すべてのファイル名はフルパスまたは相対パスで示すことができる。
- MS-DOS の attrib コマンドを使い、読み込み専用ファイルの属性を変更する。

引数に文字列式を指定してください

このステートメントには、文字列式の引数が必要です。

引数の型が一致しません

SUB, FUNCTION プロシージャのパラメータの型が、DECLARE ステートメントの引数、またはプロシージャの呼び出しに使った引数と一致しません。

解決するには次のようにしてください。

- 変数名にカーソルを合わせて [F1] キーを押し、変数のデータ型を調べて、必要に応じて変更する。
- SUB, FUNCTION の呼び出しと、それに対応する DECLARE ステートメントを比べ、定義されている引数のデータ型を確認する。

引数の個数が一致しません (ERR: 37)

SUB, FUNCTION プロシージャ呼び出しで使われている引数の数が正しくありません。

解決するには次のようにしてください。

- プログラミング環境で [検索] メニューの [検索] コマンドを使い、DECLARE ステートメントを確認する。
- SUB, FUNCTION プロシージャを定義している DECLARE ステートメントと、CALL ステートメントを比べて、引数リストの項目の数が同じかどうか確認する。

左カッコが必要です

左カッコが必要です。または、REDIM ステートメントがスカラ領域の再割り当てを行おうとしました。

ファイル モードが不正です (ERR: 54)

このエラーは、次の状況で発生します。

- 適切でないフォーマットのファイルをロードしようとした。例えば、Visual Basic for Windows のバイナリ フォーマットのファイルをロードした。
- テキストファイルをバイナリとしてロードしようとした。またはその逆。
- シーケンシャル ファイルに対して、PUT, GET ステートメントを実行した。
- ランダムアクセス モードでオープンされていないファイルに対して、FIELD ステートメントを実行した。
- 入力モードでオープンしたファイルに PRINT # ステートメントを使おうとした。
- 出力モードか追加モードでオープンしたファイルに INPUT # ステートメントを使おうとした。
- バイナリ形式で保存されているファイルを、インクルード ファイルに指定した。
- 指定したファイル番号に該当する ISAM テーブルやデータベースがない。
- CHAIN または RUN ステートメントを使い、エラーが発生した。例えば、RUN ステートメントで呼び出したプログラムに、同じ名前のプロシージャが存在する。

解決するには次のようにしてください。

- テキスト ファイルとしてロードするファイルをテキストでセーブし直す。

- テキスト ファイルとしてロードするファイルをテキストでセーブし直す。
- PUT, GET, FIELD, PRINT #, INPUT # ステートメントで、対象ファイルのモードを確認する。
- インクルード ファイルを再ロードし、テキスト形式で保存してから、プログラムを再実行する。
- すべての ISAM ファイルへの参照が適切なファイル形式に対して行うようにする。
- CHAIN または RUN ステートメントを使っている場合は、すべてのファイルをプログラミング環境にロードしてから実行し、エラー発生時のより詳しい情報を得るようにする。

ヘルプ情報を得るには、キーワードの位置にカーソルを移動して [F1] キーを押すか、キーワードをマウスの右ボタンでクリックしてください。

フォーム変数がすでに使われているか, 無効なフォーム名です

$FORM メタコマンド中のフォームの名前が、すでに変数やプロシージャの名前として宣言されているか、アルファベット以外の文字を含んでいます。

フォーム モジュールに無効なモジュールレベル コードがあります

フォーム モジュールのモジュールレベル コードに、実行可能ステートメントがあります。

次のものを除くすべてのステートメントは実行可能ステートメントです。

- COMMON
- CONST
- DATA
- DECLARE
- DEF*type*
- DIM (静的配列に対する場合)
- OPTION BASE
- OPTION EXPLICIT
- REM
- SHARED
- STATIC
- TYPE
- すべてのメタコマンド

プロシージャの中ではユーザー定義型を定義できません

ユーザー定義型の定義を、プロシージャの中で行うことはできません。

プロシージャ名が無効です

SUB, FUNCTION プロシージャの名前として、Basic の予約語や、すでに存在する変数やプロシージャの名前を使おうとしました。

プロパティ名が長すぎます

指定されたプロパティ名が 40 バイトを超えています。

変数が宣言されていません

OPTION EXPLICIT ステートメントが有効な状態で、変数を宣言せずに使おうとしました。

変数が必要です

変数の引数が必要です。

変数が必要です (ERR: 40)

コンパイル時には、変数の引数を指定しない INPUT, LET, READ, SHARED ステートメントがあると、このエラーが発生します。

実行時には、バイナリ モードでオープンされたファイルに対して、変数の引数を指定せずに GET または PUT ステートメントを実行しようとすると、このエラーが発生します。

変数名が重複しています

このエラーは次のような状況で発生します。

- 同じプログラム レベルにすでに存在する変数と同じ名前で定義しようとした。
- x という名前のユーザー定義型変数がすでに存在する場合に、SHARED ステートメントで、その名前にピリオドを付けた名前 (x.a) の変数を定義しようとした。
- x という名前のユーザー定義型変数がすでに存在し、共有されていない場合に、SUB または FUNCTION プロシージャ中で、その名前にピリオドを付けた名前 (x.a) の変数を使おうとした。

マイナス符号がありません

マイナス記号 (-) が必要です。

右カッコが必要です

右 (閉じ) カッコが必要です。

無効な .MAK ファイルです

Visual Basic for MS-DOS では無効なフォーマットの Make (.MAK) ファイルをロードしようとしました。

解決するには次のようにしてください。

- ファイル名をチェックし、内容を確認する。
- プロジェクトに必要なフォーム モジュールとコード モジュールを個々にロードする。

無効な文字があります

ソース ファイル中に制御文字などの無効な文字が見つかりました。

メタコマンドのエラーです (警告)

メタコマンドの構文に誤りがあります。メタコマンドはコメントの前に書かなければなりません。例えば、次のどちらかのようにします。

```
REM $INCLUDE: 'filespec'
' $INCLUDE: 'filespec'
```

BC コンパイラでは、このエラーは致命的ではありません。つまりプログラムを実行することはできますが、結果は保証されません。

文字列を指定してください

RSET ステートメントには、文字列の引数を指定する必要があります。

文字列変数またはユーザー定義型変数を指定してください

LSET ステートメントで、文字列変数またはユーザー定義型変数に代入できないデータ型を指定しています。数値データ型は使えません。

文字列変数を指定してください

このステートメントには、文字列変数の引数が必要です。

ユーザー定義型が定義されていません

変数または引数が、未定義のユーザー定義型で宣言されています。TYPE ステートメントを使って、ユーザー定義型を定義してください。

ユーザー定義型には BYVAL を使えません

文字列またはユーザー定義型の引数に BYVAL 節を付けて渡すことはできません。

ユーザー定義型に動的配列は使用できません

動的配列は、ユーザー定義型の要素として使うことはできません。ユーザー定義型の要素として配列を使うときは、静的配列を使ってください。

ユーザー定義型のサイズが大きすぎます

ユーザー定義型は、64K バイトを超えることはできません。複数のユーザー定義型に分割するか、要素を減らしてください。

ユーザー定義型の完了前にファイルが終りに達しました

TYPE ステートメントを解析中に、コンパイラが END TYPE を見つける前にソースファイルが終了しています。このとき、プログラムは実行を停止します。

ユーザー定義型の宣言が多すぎます

1 つのプログラムで使うことができるユーザー定義型は 240 までです。

ユーザー定義型の宣言に誤りがあるので，宣言を無視しました

TYPE ステートメントの中でエラーが起こったため、TYPE と END TYPE の間のすべてのステートメントを無視しました。

ユーザー定義型変数が必要です

ISAM の INSERT, UPDATE, RETRIEVE ステートメントを使う前には、TYPE...END TYPE ステートメントを使って、ユーザー定義型を定義する必要があります。

ユーザー定義型変数は使えません

ユーザー定義型の変数を、式の中で次のように使うことはできません。ここで、X をユーザー定義型とします。

```
CALL ALPHA((X))
```

ユーザー定義型の内容を渡したいときは、引数として、次のように要素を指定してください。

```
CALL Alpha((X.FirstEl))
```

ユーザー定義型変数名と配列変数名は使えません

ユーザー定義型の変数名と配列変数名は、READ, INPUT ステートメントの引数として使うことはできません。ただし、ユーザー定義型でない配列の要素は使うことができます。

要素が定義されていません

定義されていないユーザー定義型の要素が参照されています。例えば、ユーザー定義型の変数 MYTYPE に、要素 A, B, C が含まれているとき、変数 D を MYTYPE の要素として使おうとすると、このメッセージが表示されます。

解決するには次のようにしてください。

- TYPE...END TYPE ステートメントで、その要素が正しく定義されているかどうか確認する。
- ユーザー定義型の要素に対する参照が正しく行われているかどうか確認する。

ラベルが多すぎます

ON...GOTO, ON...GOSUB ステートメントに続く行の指定が、255 を超えています。

ラベルが重複しています (ERR: 33)

複数のプログラム行に、同じ行番号または行ラベルが割り当てられています。行番号や行ラベルは、1 つのモジュールの中で重複して付けることはできません

解決するには次のようにしてください。

- 強調表示されている項目を削除し、重複しない行番号かラベルにする。
- モジュール内のすべての SUB, FUNCTION プロシージャ定義とモジュールレベルコードをチェックして、行番号や行ラベルが重複していないか確認する。
- VBDOS プログラミング環境では、[検索] メニューの [検索] コマンドを使い、SUB, FUNCTION プロシージャの定義位置を調べる。

ラベルが未定義です (警告)

参照されている行ラベル (例えば、GOTO ステートメントのラベル) がプログラムの中に存在しません。

解決するには次のようにしてください。

- GOTO などのステートメントで参照されている行ラベルを適切な位置に挿入する。
- RUN ステートメントを使っているときは、指定されているラベルがコード モジュールのモジュールレベルの行を指していることを確認する。非実行ステートメントであれば、フォーム モジュールのモジュールレベルでもかまわない。
- ON...GOSUB と ON ERROR ステートメントがモジュールレベル コードを参照するようにする。プロシージャ レベルのエラー処理ルーチンを参照するには、ON LOCAL ERROR ステートメントを使う。

ループ カウンタの型が誤っています

FOR...NEXT 構造のループ カウンタとして、適切でない型の変数が使われていることが考えられます。FOR ループのループ カウンタは、単純数値変数でなければなりません。

D.3 LIB のエラー

Microsoft Library Manager (LIB) には、次に示すようなエラー メッセージがあります。

- 番号 U1150 から U1203 で表される致命的なエラーは、LIB の実行を中止します。
- 番号 U2152 から U2159 で表されるエラーは、実行が中止されることはありませんが、処理が不完全になる可能性があります。
- 番号 U4150 から U4158 で表される警告は、処理において生じる可能性のある問題を示します。

LIB の致命的なエラー

番号	メッセージ
U1150	page size too small; use option /PAGE:n to increase it 入力ライブラリのページサイズが小さすぎましたが、これは入力ファイル .LIB が無効であったことを示しています。
U1151	syntax error : illegal file specification マイナス記号 (-) などのコマンド演算子がありますが、その後ろにモジュール名がありません。 このエラーは、オプション指定にスラッシュ (/) のかわりにマイナス記号 (-) を使ったときに起こります。

U1152 syntax error : option name missing

スラッシュ (/) がありますが、その後ろにオプション名がありません。

U1153 syntax error : option value missing

/PAGE オプションがありますが、その後ろに値がありません。

U1154 unrecognized option

認識できないオプションが指定されています。

オプションは、スラッシュ (/) に続けて名前を指定します。名前には、正式名の正しい省略形を指定します。

U1155 syntax error : illegal input

指定されたコマンドは正しい LIB 構文になっていません。

U1156 syntax error

指定されたコマンドは正しい LIB 構文になっていません。

U1157 comma or newline missing

コマンドラインには、カンマや改行文字が必要ですが、それがありません。

以下の例のように、間違った位置にカンマを置くと、このエラーが起こります。

```
LIB math.lib, -mod1 +mod2;
```

この行の正しい入力は次のようになります。

```
LIB math.lib -mod1 +mod2;
```

U1158 terminator missing

出力ライブラリのプロンプトへの応答、または LIB の起動に使った応答ファイルの最終行の改行が抜けています。

U1059 syntax error : '}' missing in dependent

依存ファイルに対し検索パスが不正に指定されました。パスの中にスペースがあるか、閉じカッコ (}) が省略されました.

依存ファイルのディレクトリの指定の構文は, 次のようになります。

```
{directories}dependent
```

{directories} には、セミコロン (;) で区切ることにより, 複数のパスが指定できます.スペースを入れることはできません。

検索パスの一部または全てを、マクロを使うことによって置き換えることができます。マクロ展開によりスペースが入らないことを確認してください。

U1161 cannot rename old library

拡張子 .BAK のついたファイルに書き込み禁止が設定されているので、LIB は古いライブラリを拡張子 .BAK のついた名前に変更できませんでした。

古い .BAK ファイルの保護属性を変えてください。

U1162 cannot reopen library

古いライブラリは .BAK 拡張子をつけたライブラリ名に変えられた後、再オープンできませんでした。

次のような原因が考えられます。

- 別のプロセスがファイルを削除したか、読出し専用に変更されました。
- ファイルのあるフロッピー ディスクが取り出されました。
- ハードディスクにエラーが起こりました。

U1163 error writing to cross-reference file

ディスクまたはルート ディレクトリに空き領域がありません。

スペースを空けるためファイルを削除するか移動するかしてください。

U1164 name length exceeds 255 character

LIB におけるコマンドラインのファイル名の限度は 255 文字です。

名前の文字数を減らしてください。

U1170 too many symbol

すべてのオブジェクト ファイルとライブラリに現れたシンボルの数が、LIB によって作られる辞書の容量を超えています。

2 つ以上の小さなライブラリを作ってください。

U1171 insufficient memory

LIB を実行するためのメモリが不足しています。

シェルまたは常駐プログラムを削除して再試行するか、メモリを増やすかしてください。

U1172 no more virtual memory

LIB セッションは、LIB の扱える限界である 1 メガバイトを超えるメモリを必要としています。

/NOE オプションを使うか、オブジェクト モジュール数を減らしてください。

U1173 internal failure

エラーの状況を記録して、お問い合わせ先までご連絡ください。

U1174 mark : not allocated

エラーの状況を記録して、お問い合わせ先までご連絡ください。

U1175 free : not allocated

エラーの状況を記録して、お問い合わせ先までご連絡ください。

U1180 write to extract file failed

ディスクまたはルート ディレクトリに空き領域がありません。

ファイルを削除するか移動するかしてスペースを空けてください。

U1181 write to library file failed

ディスクまたはルート ディレクトリに空き領域がありません。

ファイルを削除するか移動するかしてスペースを空けてください。

U1182 *filename* : cannot create extract file

ディスクかルート ディレクトリのどちらかに空き領域がないか、あるいは指定された抽出ファイルが書込み禁止になっています。

ディスクにスペースを空けるか、書き込み禁止を解除してください。

U1183 cannot open response file

応答ファイルが見つかりません。

U1184 unexpected end-of-file on command input

プロンプトへの応答より先に EOF 文字を受取りました。

U1185 cannot create new library

ディスクかルート ディレクトリのどちらかに空き領域がないか、あるいは指定されたライブラリ ファイルは読み出し専用になっています。

ディスクにスペースを空けるか、読み出し専用属性を解除してください。

U1186 error writing to new library

ディスクまたはルート ディレクトリに空き領域がありません。

ファイルを削除または移動してスペースを空けてください。

U1187 cannot open temporary file VM.TMP

ディスクまたはルート ディレクトリに空き領域がありません。

ファイルを削除または移動してスペースを空けてください。

U1188 insufficient disk space for temporary file

ライブラリ マネジャは、仮想メモリへ書込みできません。

エラーの状況を記録して、お問い合わせ先までご連絡ください。

U1189 cannot read from temporary file

ライブラリマネジャは、仮想メモリを読み取れません。

エラーの状況を記録して、お問い合わせ先までご連絡ください。

U1190 interrupted by user

LIB が作動中に [CTRL]+[C] によって中断されました。

U1191 *libraryname* : cannot write to read-only file

指定されたライブラリは、読み出し専用に設定されているので、処理できません。

ライブラリのファイル保護属性を変更してください。

U1200 *filename* : invalid library header

指定された入力ライブラリ ファイルに無効なフォーマットがあります。

そのファイルがライブラリ ファイルではないか、または損傷しているかのどちらかです。

U1203 *filename* : invalid object file near location

filename で指定したモジュールは、有効なオブジェクト モジュールではありませんでした。

LIB のエラー

番号 メッセージ

U2152 *filename* : cannot create listing

このエラーは、次のような原因で起こります。

- ディレクトリまたはディスクに空き領域がありません。
- 書き込み禁止属性のついたクロスリファレンス リスティング ファイルが既にあります。

U2155 *module* : module not in library; ignored

指定のモジュールが入力ライブラリに見つかりませんでした。

このエラーが起こる原因のひとつに、ファイル名やディレクトリにハイフンまたはマイナス記号 (-) が含まれている場合があります。LIB は、このマイナス記号を削除コマンドだと解釈します。またこのエラーは、Microsoft 言語製品を、C:¥MS-C のようなパス名にマイナス記号を含むディレクトリにインストールしたときにも起こります。SETUP プログラムは、結合ライブラリを作るために LIB を呼び出しますが、コマンドラインにおけるマイナス記号が LIB に渡されることによって、ライブラリの構築が中断されてしまいます。

このほかに考えられる原因としては、オプション指定にマイナス記号やスラッシュを付けた場合があります。

U2157 *filename* : cannot access file

おそらくファイルがないため、LIB が指定のファイルをオープンできませんでした。

パス指定とファイル名をチェックしてください。

U2158 *library* : invalid library header; file ignored

入力ライブラリに間違ったフォーマットがあります。結合されませんでした。

U2159 *filename* : invalid format (number); file ignored

指定されたファイルは、XENIX アーカイブとして認識されなかったので、結合されませんでした。

LIB の警告

番号 メッセージ

U4150 *module* : module redefinition ignored

モジュールがライブラリへの追加を示すコマンド (+) で指定されましたが、その名前はすでにライブラリ内に存在しています。

この警告が起こる原因としては、置換えコマンド (-+) のあやまった指定が考えられます。

U4151 *symbol* : symbol defined in module module; redefinition ignored

指定されたシンボルは複数のモジュールで定義されています。

U4153 *option* : *value* : page size invalid; ignored

/PAGE オプションで指定された引数は、無効です。値は、整数の 2 乗の 16 から 32,768 の範囲でなければなりません。LIB は、結合されたライブラリの既存のページ サイズであると見なします。

U4155 *modulename* : module not in library

コマンドで指定されたモジュールは、ライブラリに存在しません。

ライブラリに対象のモジュールがなく、置換えコマンド (-+) が指定されると、LIB はファイルを追加します。また、削除コマンド (-) または移動コマンド (-*) が指定されたときには、LIB はそのコマンドを無視します。

U4156 *library* : output-library specification ignored

library フィールドで指定されたファイルが存在しないので、新しいライブラリが作られました。ここで、同じファイルが *newlibrary* フィールドで指定されていると、LIB は *newlibrary* の指定を無視します。

たとえば、次に示すコマンドラインは、PROJECT.LIB が存在しないときにこの警告を起こします。

```
LIB project.lib +one.obj, new.lst, project.lib
LIB project.lib +one.obj, new.lst, new.lib
```

U4157 insufficient memory, extended dictionary not created

メモリが不足しているので LIB は拡張辞書を作れません。

ライブラリは有効ですが、リンカはリンクのスピードアップに拡張辞書を利用することができません。

U4158 internal error, extended dictionary not created

内部エラーのため LIB は拡張辞書を作れません。

ライブラリは有効ですが、リンカはリンクのスピードアップに拡張辞書を利用することができません。

D.4 LINK のエラー

Microsoft Segmented Executable Linker (LINK) には、次に示すようなエラー メッセージがあります。

- 番号 L1*xxx* は、致命的なエラーです。致命的なエラーが発生すると LINK の実行を中止します。
- 番号 L2*xxx* が発生すると、LINK の実行は中止されませんが、出力ファイルは作成されません。
- 番号 L4*xxx* で表される警告 エラーは、出力ファイルの作成において生じる可能性のある問題を示します。

LINK の致命的なエラー

番号 メッセージ

L1001 *option* : option name ambiguous

オプション記号 (/) の後に指定されたオプション名が識別できません。

オプションの指定は、スラッシュ (/) の後にオプション名をつけます。オプション名は省略名でかまいませんが、明確に指定してください。

たとえば、オプション名の多くは文字 N から始まるので、次のようなコマンドはこのエラーの原因になります (省略しすぎです)。

```
LINK /N main;
```

このエラーはまた、異なったバージョンのリンカを使うことによっても発生します。環境変数 PATH のディレクトリを確認して、別のバージョンの LINK.EXE を起動していないか確認してください。

L1003 /Q and /EXEPACK incompatible

LINK では、/Q オプションと /EXEPACK オプションを同時に指定することはできません。

L1004 *value* : invalid numeric value

LINK オプションの 1 つにあやまった値が指定されています。

たとえば、数値が必要なオプションに対して文字列が与えられています。

L1005 *option* : packing limit exceeds 64K

/PACKCODE または /PACKDATA オプションで与えられた値が 65,536 バイトの限度を超えています。

L1006 *number* : stack size exceeds 64K-2

次に示すような原因が考えられます。

- /STACK オプションにパラメータとして与えられた値が最大限度 (65,535 バイト) を超えています。
- /STACK オプションとその引数の間にあるコロン (:) の前後に、不要なスペースがあります。

L1007 /OVERLAYINTERRUPT : interrupt number exceeds 255

255 より大きい数値が /OVERLAYINTERRUPT のオプション値として指定されています。

MS-DOS のマニュアルを参照して、割り込みの解説を調べてください。

L1008 /SEGMENTS : segment limit set too high

/SEGMENTS オプションで指定された論理セグメントの最大数が 16,375 を超えています。

L1009 *value* : /CPARM : illegal value

/CPARM オプションで指定された値が 1～65,535 の範囲に入っていません。

L1020 no object files specified

リンカにオブジェクトファイル名が指定されていません。

リンカは最低でも 1 つのオブジェクトファイル名の指定を必要とします。

L1021 cannot nest response files

応答ファイルの中に応答ファイルの呼出しがあります。

L1022 response line too long

応答ファイルの中に 255 字を超える行があります。

行を分割するには、分割する行の終わりにプラス記号 (+) を入力します。

L1023 terminated by user

[CTRL]+[C] キーが入力されました。

L1024 nested right parenthesis

コマンド行のオーバーレイ指定の内容にミスタイプがあります。

L1025 nested left parenthesis

コマンド行のオーバーレイ指定の内容にミスタイプがあります。

L1026 unmatched right parenthesis

コマンド行のオーバーレイの指定で左かっこが抜けています。

L1027 unmatched left parenthesis

コマンド行のオーバーレイの指定で右かっこが抜けています。

L1030 missing internal name

IMPORTS 文にモジュール定義ファイル中の序数の指定がありますが、インポートするルーチンまたはデータの内部名が抜けています。

インポートを序数で行うときは内部名が必要です。

L1031 module description redefined

モジュール定義 (.DEF) ファイル中の DESCRIPTION 文の指定が重複しています。

L1032 module name redefined

NAME または LIBRARY モジュール名が重複して定義されています。

L1033 input line too long; *number* characters allowed

LINK のコマンドラインは、*number*で示された文字数を超えて指定することはできません。

L1034 name truncated to *string*

LINK のコマンドライン、またはモジュール定義 (.DEF) ファイル で指定された名前は、255 文字を超えています。名前は *string* に切り詰められました。

このエラーは警告で、致命的なエラーではありませんが、重大な問題を示しています。このメッセージは、LINK が指定された名前を使うときの、ほかのエラーといっしょに出力されることがあります。たとえば、その名前がファイル名だった場合、LINK がそのファイルをオープンできなくなり、エラーを起こします。

L1035 syntax error in module-definition file

モジュール定義 (.DEF) ファイル 内のステートメントにあやまりがあります。

L1040 too many exported entries

プログラム中のエクスポート名の数が、限度数 65,535 を超えました。

L1041 resident names table overflow

常駐名テーブルのサイズが 65,535 バイトを超えました。

RESIDENTNAME と表示された各エクスポートルーチンごとに、常駐テーブルに名前の 3 バイトの情報が入力されます。最初のエントリはモジュール名です。

エクスポートルーチンの数を減らすか、非常駐に変えてください。

L1042 nonresident names table overflow

非常駐名テーブルのサイズが 65,535 バイトを超えました。

RESIDENTNAME と表示されなかった各エクスポートルーチンごとに、非常駐テーブルに名前と 3 バイトの情報が入力されます。最初のエントリは DESCRIPTION 文です。

エクスポートルーチンの数を減らすか、一部を常駐に変えてください。

L1043 relocation table overflow

プログラム内の long 呼び出し、long ジャンプその他の long ポインタの数が 32,768 を超えました。

long 参照を short 参照に変えることのできるところでは変えてください。

L1044 imported names table overflow

インポート名テーブルのサイズが 65,535 バイトを超えています。

IMPORTS 部分に入る新しい名前ごとに、インポートテーブルにモジュール名とインポート名プラス 1 バイトが入力されます。

インポート数を減らしてください。

L1045 too many TYPDEF records

オブジェクト ファイルに入っている TYPDEF レコードが 255 個を超えています。

このレコードは共用変数を示しています (TYPDEF は MS-DOS 用語です。この説明は MS-DOS のマニュアルや、その他 MS-DOS 関連の参考書に記載されています)。

このエラーは、Microsoft FORTRAN コンパイラ、または共用変数をサポートするほかのコンパイラで作られたプログラムでのみ発生します。

L1046 too many external symbols in one module

オブジェクト ファイルに指定された外部シンボルが限度の 1,023 を超えています。

オブジェクト ファイルを小さく分割してください。

L1047 too many group, segment, and class names in one module

プログラム中のグループ、セグメント、クラス名の数が多すぎます。

オブジェクト ファイル中のグループ、セグメント、クラスの数を減すか、オブジェクトファイルを小さく分割してください。

L1048 too many segments in one module

オブジェクト ファイルに 255 以上のセグメントがあります。

モジュールを分割するかセグメントをまとめます。

L1049 too many segments

プログラム中のセグメント数が限度を超えています。

リンク時に /SEG オプションを使って、セグメントの有効な最大数を指定してください。有効範囲は 1～16,375 です。デフォルトは 128 です。

p-code プログラムをリンクするときにこのエラーが発生したときは、テンポラリな p-code セグメントを結合するために、/NQ オプションを指定して再コンパイルしてください。

L1050 too many groups in one module

LINK 処理中に、1 つのオブジェクト ファイル中に 21 を超えるグループ定義 (GRPDEF) が見つかりました。

グループ定義の数を減らすか、モジュールを分割してください。

(グループ定義の説明は、MS-DOSのマニュアルを参照してください。)

L1051 too many groups

プログラムに DGROUP 以外に、20 以上のグループが定義されています。

グループ数を減らしてください。

L1052 too many libraries

32 を超えるライブラリをリンクしようとしました。

ライブラリをまとめるか、少ないライブラリ数で済むモジュールを使ってください。

L1053 out of memory for symbol table

プログラムに使用可能なメモリ量を超えるシンボル情報 (パブリック、外部、セグメント、グループ、クラス、ファイル名など) があります。

次に示す対応策が考えられます。

- できるだけパブリック シンボルの数を減らしてください。
- オブジェクト ファイルやセグメントをまとめてください。
- リンクを、コマンドラインから実行してください。
- メモリ常駐 (TSR) プログラムの常駐を解除するか、あるいは何らかの方法で空きメモリを増やしてください。

L1054 requested segment limit too high

LINK には要求されたセグメント数を記述するテーブルに割り当てるだけのメモリがありません。セグメント数はデフォルトの 128 か、/SEGMENTS オプションで指定した値のどちらかです。

次に示す対応策が考えられます。

- セグメント数を減らすために /SEG オプションを指定して再リンクしてください。
- メモリ常駐 (TSR) プログラムの常駐を解除するか、あるいは何らかの方法で空きメモリを増やしてください。

L1056 too many overlays

プログラムで定義したオーバーレイが 127 を超えています。

L1057 data record too large

あるオブジェクトモジュールの中の LEDATA レコードに、1,024 バイト以上のデータが入っています。これは翻訳プログラムのエラーです (LEDATA については、MS-DOS のマニュアルを参照してください)。

あやまったモジュールがどの翻訳プログラム (コンパイラかアセンブラ) から出たのかを調べて下さい。エラーの状況をパッケージに入っている問い合わせ用紙に記入して、Microsoft までお送りください。

L1063 out of memory for debugging information

デバッグ情報のあるオブジェクトファイルの数が多くなりすぎて、LINK が格納するスペースがなくなりました。

一部のファイルを、/Zi の代わりに /Zd を使ってコンパイルするか、CodeView オプションを使わずにコンパイルし、完全なデバッグ情報の入ったオブジェクト ファイルの数を減らしてください。

L1064 out of memory - near/far heap exhausted

十分なメモリがないために、LINK が 'near' または 'far' ヒープにメモリを割り当てることができません。

次に示す対応策が考えられます。

- プログラム中のコード、データ、シンボルのサイズを小さくしてください。
- プログラムが .EXE ファイルならば、コードの一部をダイナミックリンク ライブラリに格納してください。

L1065 too many interoverlay calls use /DYNAMIC:*num*; current limit is *number*

プログラム中の内部オーバーレイ コールが多すぎます。

内部オーバーレイ コールの最大値は、/DYNAMIC オプションで指定します (有効範囲は 1～10,922 です)。/DYNAMIC オプションが省略されたときのデフォルトは 256 です。

プログラムに必要な設定を知るには、LINK を /INFO オプションを指定して実行します。現在の内部オーバーレイ コールの数や、現在の限界値を知ることができます。

L1066 size of *overlaynumber* overlay exceeds 64K

与えられた値によって示されるオーバーレイは、MOVE サイズの限界値である 65,535 バイトを超えています。

L1067 memory allocation error

エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1070 *segment* : segment size exceeds 64K

1 つのセグメントの中のコードまたはデータが 65,536 バイトを超えています。

メモリモデルを変えて far コードまたはデータを適切に使うようにしてください。C 言語プログラムならば、CL の /NT オプションか __based キーワード (または alloc_text を使って、もっと小さなセグメントをビルドしてください。

L1071 segments _TEXT exceeds 64K-16

_TEXT という名前のセグメント サイズが 65,520 を超えてしまいました。このエラーが発生するのは、スモールモデルの C プログラムだけではなく、_TEXT という名のセグメントのあるプログラムで、LINK コマンドの /DOSSEG オプションを指定してリンクすると、発生することがあります。

スモールモデルの C プログラムでは、コードアドレス 0 と 1 を必ず予約しておいてください。この範囲はアラインメントの目的によって 16 に増えます。

ミディアムまたはラージモデルを使ってコンパイルまたはリンクしてみてください。C 言語プログラムならば、CL の /NT オプションか__based キーワード (または alloc_text プラグマ)を使って、もっと小さなセグメントをビルドしてください。

L1072 common area exceeds 64K

プログラムに 65,536 バイト以上の共用変数があります。このエラーは、Microsoft FORTRAN コンパイラ、または共用変数をサポートする他のコンパイラで作られたプログラムでのみ発生します。

L1073 file-segment limit exceeded

物理セグメントまたはファイルセグメントの数が、Windows において、各アプリケーションまたはダイナミックリンク ライブラリにおける限度の 255 を超えました。

ファイルセグメントは、各グループ定義、パックされてない論理セグメント、パックされたセグメントのセットごとに作られます。

セグメント数を減らすか、各セグメントごとの情報量をもっと増やしてください。/PACKC オプションと /PACKD オプションのどちらか片方 (または両方) が指定してあることを確認してください。

L1074 *group* : group exceeds 64K

表示されたグループは限度の 65,535 バイトを超えています。

グループのサイズを小さくするか、グループから不要なセグメントを除いてください。セグメントの一覧はマップファイルで参照してください。

L1075 entry table exceeds 64K-1

エントリテーブルの大きさが限度の 65,535 バイトを超えています。

このテーブルには、PROTMODE が有効ではなく、ターゲットセグメントが MOVABLE になっているときに、エクスポートした各ルーチンごとのエントリと、far リロケーションのターゲットアドレスごとのエントリが入ります。

PROTMODE が使えれば宣言します。あるいはエクスポートしたルーチンの数を減らすか、可能ならばセグメントの一部を FIXED にしてください。

L1078 file-segment alignment too small

/ALIGNオプションで指定したセグメントアラインメントのサイズが小さすぎます。数を増やしてください。

L1080 cannot open list file

ディスクまたはルートディレクトリに空き領域がありません。

ファイルを消去するか移動してスペースを空けてください。

L1081 out of space for run file

ディスクまたはルートディレクトリに空き領域がありません。

ファイルを消去するか移動してスペースを空けてください。

L1082 *filename* : stub file not found

LINK は、モジュール定義 (.DEF) ファイルの STUB 命令文で指定されたファイルをオープンできませんでした。

モジュール定義 (.DEF) ファイルは、カレントディレクトリか、PATH 環境変数で指定されたディレクトリのどちらかに入っていなければなりません。

L1083 cannot open run file

次のような原因が考えられます。

- ディスクまたはルートディレクトリに空き領域がありません。
- 別のプロセスがそのファイルをオープンまたは削除しました。
- 同じ名前の読み出し専用ファイルがあります。
- そのファイルが存在しているフロッピーディスクがはずされました。
- ハードディスクにエラーが発生しました。

L1084 cannot create temporary file

次のような原因が考えられます。

- ディスクかルートディレクトリに空き領域がありません。
- 環境変数 TMP で指定されたディレクトリは存在しません。

L1085 cannot open temporary file - *message*

LINK は、*message* に示された理由によってテンポラリ ファイルをオープンできませんでした。

次のような原因が考えられます。

- ディスクかルートディレクトリに空き領域がありません。
- 環境変数 TMP で指定されたディレクトリは存在しません。

L1086 temporary file missing

内部エラーが起きました。

エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1087 unexpected end-of-file on temporary file

リンカの出力するテンポラリ ファイルに問題が発生しました。

次のような原因が考えられます。

- テンポラリ ファイルのあるディスクがはずされました。
- 環境変数 TMP に指定されているディスクまたはディレクトリに空き領域がありません。

L1088 out of space for list file

ディスクまたはディレクトリに空き領域がありません。

ファイルを削除または移動させて、スペースを空けてください。

L1089 *filename* : cannot open response file

リンカが指定の応答ファイルを見つけられませんでした。

次のような原因が考えられます。

- 応答ファイルが存在しません。
- 応答ファイル名の指定にあやまりがあります。
- 古いバージョンの LINK を使っています。パスをチェックしてください。LINK のバージョン番号を確認するには、LINK に /? オプションを指定して実行します。

L1090 cannot reopen list file

オリジナル ディスクがプロンプトに従って差し替えられていません。

リンカを再実行させてください。

L1091 unexpected end-of-file on library

ライブラリの入っているディスクがはずされました。

ライブラリの入っているディスクに差し替えてリンカを再実行させてください。

L1092 cannot open module-definition file

LINK は、指定されたモジュール定義 (.DEF) ファイル が見つかりませんでした。

.DEF ファイル名が正しく指定されているか確認してください。

L1093 *filename* : object file not found

LINK は、指定されたオブジェクト ファイルが見つかりませんでした。

オブジェクト ファイル名が正しく指定されているか確認してください。

L1094 *filename* : cannot open file for writing

LINK は、指定されたファイルを書き込み可能なファイルとしてオープンできませんでした。

ファイルの属性を確認してください。

L1095 *filename* : out of space for file

LINK に指定した出力ファイル用のディスク スペースがなくなりました。

ファイルを消去するか移動して空き領域を広げてください。

L1096 unexpected end-of-file in response file

応答ファイルの読み込み中に問題が発生しました。

次のような原因が考えられます。

- 応答ファイルが壊れています。
- 応答ファイルは読み込み中に削除されました。

L1097 I/O error - *message*

入出力エラーが発生しました。

L1098 cannot open include file *filename* - *message*

LINK は、*message*に示された理由によって、指定されたインクルード ファイルをオープンできませんでした。

インクルード ファイルは、モジュール定義 (.DEF) ファイルの INCLUDE ステートメントで指定します。

L1100 stub .EXE file invalid

STUB 文に指定されているファイルは、MS-DOS の有効な実行形式ファイルではありません。

L1101 invalid object module

LINK は、オブジェクト モジュールの 1 つをリンクできません。

リンクに使った LINK のバージョンが正しいかどうか確認してください。

再コンパイル後もエラーが消えないときは、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1102 unexpected end-of-file

指定されたライブラリまたはオブジェクト ファイルのフォーマットは無効です。

L1103 attempt to access data outside segment bounds

オブジェクト ファイル内のデータ レコードに、セグメントの末尾を越えるデータの指定があります。これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1104 *filename* : invalid library

指定したライブラリのフォーマットは無効です。

L1105 invalid object due to interrupted incremental compile

オブジェクト ファイルを消去して、プログラムを再コンパイル、再リンクしてください。

L1106 unknown COMDAT allocation type for *symbol*; record ignored

これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1107 unknown CONDAT selection type for *symbol*; record ignored

これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1108 invalid format of debugging information

これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1113 unresolved COMDEF; internal error

これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1114 unresolved COMDAT *symbol*; internal error

これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1115 *option* : option incompatible with overlay

指定されたオプションは、オーバーレイされたプログラムのリンクには使えません。

L1117 ununallocated COMDAT *symbol*; internal error

これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L1123 *segment* : segment defined both 16-bit and 32-bit

セグメントを 16 ビットか 32 ビットのどちらかで定義してください。

L1127 far segment references not allowed with /TINY

.COM ファイルを作るための /TINY オプションが、far セグメントを参照するモジュールで使われています。

far セグメント参照は、.COM ファイル フォーマットと適合しません。言語が高水準言語であり、スモール メモリ モデルをサポートしていない場合、このエラーが起こります。セグメント アドレスを参照するアセンブリ言語のプログラムでもこのエラーが起こります。

たとえば、次のプログラムはこのエラーを発生します。

```
mov     ax, seg mydata
```

L1128 too many nested include files in module-definition file

モジュール定義 (.DEF) ファイル内の INCLUDE ステートメントのネストが、10 レベルを超えています。

L1129 missing or invalid include file name

モジュール定義 (.DEF) ファイル内の INCLUDE ステートメントで指定したファイル名にあやまりがあります。

LINK のエラー

番号	メッセージ
L2000	imported starting address アセンブリ言語ファイルの END 文で指定されたプログラム開始アドレスが、インポート ルーチンです。これは Windows ではサポートされません。
L2002	fixup overflow at *number* in segment *segment* このエラーメッセージには次のどちらかの表示が続きます。 ■ target external *symbol* ■ frm seg *name1*, tgt seg *name2*, tgt offset *number* フィックスアップ オーバーフローとは、ソース位置 (参照が起きる“元”) とターゲット アドレス (参照する“先”) が離れすぎているので、コードやデータの参照ができないことをいいます。通常、この問題はソース中の位置を調べて解決することができます。 ソース ファイルを修正し、オブジェクト ファイルをつくり直してください。フレームとターゲット セグメントについては、MS-DOS のマニュアルを参照してください。
L2003	near reference to far target at *offset* in segment *segment* pos: *offset* target external *name* プログラム内に、別のセグメント中のラベルへの near コールまたは near ジャンプが記述されています。 このエラーが一番よく発生するのは、far として宣言しなければならない外部プロシージャを near プロシージャとして宣言したときです。 このエラーは、スモールモデルの C プログラム を CL の /NT オプションでコンパイルしたときにも発生します。
L2005	fixup type unsupported at *number* in segment *segement* LINK がサポートしていない フィックスアップ タイプが起きました。コンパイラのエラーです。 エラーの状況を記録して、お問い合わせ先までご連絡ください。
L2010	too many fixups in LIDATA record LIDATA での (ポインタまたはベースタイプの) far 再配置の数が、LINK の設定している限度を超えました。 このエラーが発生する典型的な例は、アセンブリ言語ファイルの DUP 文です。再配置と LIDATA レコードで 1024バイトのバッファを重複して使っています。再配置では、8 バイトを使います。 DUP 文の far 再配置の数を減らしてください。

L2011 *identifier* : NEAR/HUGE conflict

共用変数へ与えられている NEAR と HUGE の属性が、衝突しています。このエラーは、Microsoft FORTRAN コンパイラか他の共用変数をサポートするコンパイラのみで発生します。

L2012 *arrayname* : array-element size mismatch

1 つの far 共用配列が 2 つ以上の違った配列要素サイズで宣言されています (たとえば配列が、文字の配列と実数の配列として宣言されています)。このエラーが起こるのは、Microsoft FORTRAN コンパイラや far 共用変数をサポートするコンパイラのときだけです。

L2013 LIDATA record too large

LIDATA レコードに 512 バイト以上のデータがあります。トランスレータのエラーです。

エラーの状況を記録して、お問い合わせ先までご連絡ください。

L2022 *entry* (alias *internalname*) : export undefined

指定されたエクスポート ルーチンの内部名が定義されていません。

L2023 *entry* (alias *internalname*) : export imported

指定されたエクスポート ルーチンまたはデータの内部名が、前にインポートしたルーチンまたはデータの内部名と重複しています。

L2024 *symbol* : special symbol already defined

プログラムが定義したシンボル名は、すでに LINK が、それ自身の下位レベル シンボルとして使っています。たとえば LINK は、オーバーレイ サポートや他の操作で使われる特別のシンボルを生成します。

重複を避けるため別のシンボル名を選んでください。

L2025 *symbol* : symbol defined more than once

同じシンボル名が 2 つ以上のオブジェクト ファイルで見つかりました。

L2026 entry ordinal *number*, name *name* : multiple definitions for same ordinal

指定された順序数のエクスポート名は、既に同じ順序数につけられた別のエクスポート名と衝突します。1 つの順序数には 1 つの名しかつけられません。

L2027 *name* : ordinal too large for export

指定されたエクスポート名は、限度 65,535 (64k-1) を超える順序数に割り当てられています。

L2028 automatic data segment plus heap exceed 64K

次に示す項目の和が 64K を超えています。

- DGROUP に宣言されたデータ
- モジュール定義ファイルの HEAPSIZE で指定されたヒープ サイズ
- /STACK オプションまたは STACKSIZE モジュール定義 (.DEF) ファイル文で指定されたスタック サイズ

near データ アロケーション、HEAPSIZE、スタックなどを減らしてください。

L2029 *symbol* : unresolved external

1 つまたは複数のモジュールで、あるシンボルが外部シンボルとして宣言されていますが、どのモジュールにもライブラリにもまだ定義されていません。

指定されたの未定義の外部シンボル名が先ず示され、次にこのシンボルへのリファレンスの入ったオブジェクト モジュールのリストが示されます。このメッセージとリストは、マップ ファイルがあればそれに書き込まれます。

このエラーは、/NOI オプションを使った場合に起こることがあります。

また、Microsoft C/C++ バージョン 7.0 で コンパイルしたプログラムを、/NOD オプション付きでリンクした場合にも起こります。LINK は/NOD オプションが指定されると、オブジェクト ファイル内のデフォルトのライブラリ名を無視します。Microsoft C/C++ バージョン 7.0 は、オブジェクト ファイルに、デフォルトのランタイム ライブラリ名と OLDNAMES.LIB の両方を埋め込みます。このエラーを回避するには、*libraries* フィールドに OLDNAMES.LIB を指定するか、/NOD:*libraries* を指定します。*libraries* は、検索時のデフォルトのランタイム ライブラリの名前です。

L2030 starting address not code (use class 'CODE')

プログラムの開始アドレスは、.ASM ファイルの END 文での指定と同様、コード セグメントになっていなければなりません。コード セグメントはクラス名の最後が "CODE" になっていれば認識されます。このエラーは、セグメント化された実行可能プログラムで発生することがあります。

モジュール定義 (.DEF) ファイルに REALMODE 文を入れると、このエラー メッセージを無効にすることができます。

L2041 stack plus data exceed 64K

near データと要求したスタックのサイズ合計が 64K を超えると、プログラムは正しく動作しません。リンカがこの状態をチェックするのは、/DOSSEG を指定しているときだけです。例として、ライブラリのスタートアップモジュールがあります。

Microsoft C または Microsoft FORTRAN でコンパイルされたオブジェクト モジュールに関しては、データ サイズをより小さな値にするように、/Gt コマンドライン オプションで再コンパイルしてください。

これは致命的な LINK エラーです。

L2043 Quick library support module missing

必要なライブラリ VBDOSQLB.LIB がありません。Quick ライブラリを作るときは VBDOSQLB.LIB モジュールをリンクしなければなりません。

L2044 *symbol* : symbol multiply defined, use /NOE

LINK は、パブリック シンボルの再定義とみなされる状態を見つけました。原因は、おそらくライブラリで定義されたシンボルの重複定義です。

/NOE オプションでリンクし直してください。同じシンボルで L2025 エラーが起こるようならば、本当のシンボル再定義エラーです。

L2046 share attribute conflict - segment *segment* in group *group*

*segment*で示されたセグメントは、*group* で示されたグループに割り当てられた別のセグメントと、異なる共有属性を持っています。異なった属性は指定できません。

すべてのセグメントは、SHARED や NONSHARED のどちらかを使って、グループに対して同じ属性を持つようにします。

L2047 IOPL attribute conflict - segment *segment* in group *group*

指定のセグメントは指定のグループに属していますが、そのグループの中の他のセグメントとは異なる IOPL 属性を持っています。

L2048 Microsoft Overlay Manager module not found

オーバーレイが指示されているのに、オーバーレイ マネージャが見つかりません。

デフォルトでは、オーバーレイ マネージャは、Microsoft Overlay Virtual Environment (MOVE) です。これは、Microsoft C/C++ バージョン 7.0 に含まれるデフォルトの結合されたライブラリの、コンポーネント ライブラリ MOVE.LIB で提供されます。このエラーは、LINK が _moveinit ルーチンを見つけられなくなったときも起こります。

/OLDOVERLAY オプションが指定されると、オーバーレイ マネージャは、デフォルトの結合ライブラリから提供される Microsoft Static Overlay Manager となります。

L2050 USE16/USE32 attribute conflict - segment *segment* in group *group*

16 ビット セグメントは、32 ビット セグメントと同じグループには入れられません。

L2052 *symbol* : unresolved external; possible calling convention mismatch

1 つ以上のモジュールで外部宣言されているシンボルがありますが、そのシンボルがどのモジュールやライブラリにもパブリック定義されていないので、LINK が発見できません。

その未決外部参照のシンボル名が与えられ、このシンボルへの参照を含むオブジェクト モジュールのリストが与えられています。エラー メッセージとこのリストは、マップ ファイルがあれば、そこに書き込まれています。

このエラーは、外部で定義された関数のプロトタイプが制限され、fast call オプション (/Gr) 付きでコンパイルされたに発生します。__fastcall の呼び出し規約は、外部関数のためのプロトタイプがインクルードされることを仮定していません。

その関数用にプロトタイプをインクルードするか、/Gr オプションなしでコンパイルしてください。

L2057 duplicate of *function* with different size found; record ignored
矛盾するクラスの定義が見つかりました。

インクルード ファイルをチェックして、再コンパイルしてください。

L2058 different duplicate of *function* found; record ignored
矛盾するクラスの定義が見つかりました。

インクルード ファイルをチェックして、再コンパイルしてください。

L2060 size of data block associated with *symbol* (16-bit segment) sxceeds 64K
クラスに、仮想関数が多すぎます。指定されたシンボルは、クラスに対する v-table で、装飾名の形式です。

L2061 no space for data block associated with *function* inside *segment* segment
指定された関数は、指定さられたセグメントに割り当てられていますが、そのセグメントには空き領域がありません。

L2062 continuation of COMDAT *function* has conflicting attributes; record ignored
これはトランスレータのエラーです。

どのトランスレータ プログラム (コンパイラまたはアセンブラ) であやまったオブジェクト モジュールができたか、またどのような状況でできたかを調べてください。また、エラーの状況を記録して、お問い合わせ先までご連絡ください。

L2063 *function* is allocated in undefined segment
指定された関数は、存在しないセグメントに割り当てようとしています。

L2064 starting address not in the root overlay
プログラムのスタート アドレスを含むセグメントまたはオブジェクト ファイルが、オーバーレイの中に存在します。

C 言語のプログラムでは、スタート アドレスは main 関数の中になければなりません。

LINK の警告

番号 メッセージ

L4000 segment displacement included near *offset* in segment *segment*
これは /W オプションが出す警告です。

L4001 frame-relative fixup, frame ignored near *offset* in segment *segment*
参照は、その参照のターゲット セグメントとは違うセグメントに関連しています。

たとえば、セグメント `_TEXT` に `_id1` が定義されているとき、命令 `DGROUP:_id1` が、この警告を出します。フレーム `DGROUP` は無視されるので、LINK は、`_TEXT:_id1` としてこの命令を扱います。

L4002 frame-relative absolute fixup near *offset* in segment *segment*

参照は、その参照のターゲット セグメントとは違うセグメントとの相対とされていますが、双方のセグメントは絶対になっています (ATで定義)。

LINK は、その実行形式ファイルをリアル モードのみで実行するものとして扱います。

L4004 possible fixup overflow at *offset* in segment *segment*

near コールか near ジャンプが、他のセグメントに行われましたが、そのセグメントは、元のセグメントとは別のグループに属しています。

このコールやジャンプにより、セグメント グループのパラグラフ アドレス (フレーム番号) とターゲット セグメントとの距離が 64K 以上あると、たとえ元のセグメントとターゲット セグメント間の距離が 64K 以下であっても、リアル モードでのアドレス計算に間違いが生じることがあります。

L4010 invalid alignment specification

/ALIGN オプションで指定する数は、2 から 32,768 までの範囲の 2 の累乗でなければなりません。

L4011 /PACKC value exceeding 64K-36 unreliable

/PACKC オプションで指定されたパッキングの限度が 65,501 から 65,536 バイトの間でした。この範囲に入るサイズのコード セグメントは、CPU が 80286 であると、信頼性が保証できないことがあります。

L4012 /HIGH disables /EXEPACK

/HIGH オプションと /EXEPACK オプションは同時に使えません。

L4013 *option* : option ignored for segmented executable file

指定されたオプションは、セグメント化された実行可能プログラムでは使えません。

L4014 *option* : option ignored for DOS executable file

指定されたオプションは、MS-DOS プログラムでは使えません。

L4015 /CO disables /DSALLOC

/CO オプションと /DSALLOC オプションは、同時に使えません。

L4016 /CO disables /EXEPACK

/CO オプションと /EXEPACK オプションは、同時に使えません。

L4017 *option* : unrecognized option name; option ignored

オプションに認識できない文字があります。LINK はそのオプションを無視します。

次のような原因が考えられます。

- 旧バージョンのオプションが、現在のバージョンの LINK に指定されています。たとえば、/INCR オプションが LINK バージョン 5.30 に指定されるような場合です。現在サポートされるオプションは、マニュアルまたはオンライン ヘルプで知ることができます。また、/? オプションを指定して LINK を実行すると、オプションのリストが表示されます。

- 旧バージョンの LINK を使っています。パスを確認してください。LINK のバージョン番号を知るには、/? オプションを指定して LINK を実行します。
- オプション名にあやまりがあります。たとえば、/NODEFAULTLIBSEARCH は、/NODEFAULTLIBRARYSEARCH の省略形ではありません。オプション名の省略形は、正式名称の末尾を省略する形になっています。

L4018 missing or unrecognized application type; option *option* ignored

/PM オプションが受け付けるのは、PM, VIO, NOVIO だけです。

L4020 *segment* : code-segment size exceeds 64K-36

65,501 から 65,536 バイト長までのコード セグメントは、CPU が 80286 であると信頼性の低いことがあります。

L4021 no stack segment

プログラムに STACK コンバイン タイプで定義したスタック セグメントがありません。

通常、どのプログラムにも、STACK として指定されたコンバイン タイプのスタック セグメントが必要です。しかし、スタックを定義しないか、あるいは STACK コンバイン タイプなしで定義する特別の理由があるならば、このメッセージを無視してもかまいません。バージョン 2.40 以前のバージョンの LINK でリンクした場合には、リンカがライブラリを一度しかサーチしないので、このメッセージが表示されることがあります。

L4022 *group1*, *group2* : group overlap

指定されたグループがオーバーラップしています。1 グループは 1 物理セグメントに割り当てられるので、セグメント化された実行可能プログラムでは、グループのオーバーラップは許されません。

グループがオーバーラップしないよう、セグメントとグループ定義を組み直してください。マップファイルを参照してください。

L4023 *entry* (*internalname*) : export internal name conflict

指定されたエクスポート ルーチンの内部名が、以前のインポート定義またはエクスポート定義と重複しています。

L4024 *name* : multiple definitions for export name

このエクスポート名は 2 回以上エクスポートされていますが、2 回以上エクスポートできません。

L4025 *modulename.entry* (*internalname*) : import internal name conflict

インポート ルーチン (インポートは名前か番号) の内部名が、以前のエクスポート、またはインポートの内部名と重複しています。

L4026 modulename.entry (internalname) : self-imported

指定された関数またはデータ項目は、リンク中のモジュールからインポートされています。このエラーは、モジュールが、それ自身または同じ名前の別のソースから関数やデータ項目をインポートしようとすると起こります。

L4027 *name* : multiple definitions for import internal name

内部名が 2 回以上インポートされています。前のインポート定義は無視されます。

L4028 *segment* : segment already defined

指定されたセグメントは、モジュール定義 (.DEF) ファイル の SEGMENTS 文で 2 回以上定義されました。

L4029 *segment* : DGROUP segment converted to type DATA

グループ DGROUP の指定された論理セグメントが、コード セグメントとして定義されました。

LINK は DGROUP を、常にデータ セグメントとして扱っているので、DGROUP にはコード セグメントを入れることはできません。DGROUP という名は自動またはデフォルトデータ セグメントとして前もって定義されています。

LINK は命名したセグメントをタイプ DATA に変更します。

L4030 *segment* : segment attributes changed to conform with automatic data segment

DGROUP の中で与えられた論理セグメントは、DATA として宣言された DATA 自動データ属性 (SINGLE/MULTIPLE) とは違う共有属性 (SHARED/NONSHARED) を与えられています。この属性は、DGROUP の属性に合わせて変換されます。

DGROUP という名は自動またはデフォルト データ セグメントとして前もって定義されています。LINK は DGROUP を常にデータ セグメントとして扱っているので、DGROUP にはコード セグメントを入れることはできません。

L4031 *segment* : segment declared in more than one group

1 つのセグメントを、異なる 2 つのグループのメンバーとして宣言しました。

L4032 *segment* : code-group size exceeds 64K-36

65,501 から 65,536 バイト長までのコード セグメントは、CPU が 80286 であると信頼性の低いことがあります。

L4033 first segment in mixed group *group* is a USE32 segment

16 ビットのセグメントは、USE16 と USE32 のセグメントの両方で作られたグループの最初になければなりません。

LINK は実行形式ファイルの作成を続行しますが、その結果出力されたファイルは正常に動作しない可能性があります。

L4034 more than 1024 overlay segments; extra put in root

オーバーレイに入れられるセグメントの数は最高 1,024 までです。1,025 番目のセグメントはプログラムの常駐部分 (ルート) に配置されます。

L4036 no automatic data segment

アプリケーションが DGROUP という名のグループを定義していません。

DGROUPという名は、オペレーティング システムが使う自動またはデフォルト データ セグメントの識別に使われるので、リンカにとって特別の意味を持っています。ほとんどの セグメント化された実行可能プログラムのアプリケーションには DGROUP が必要です。

DATA NONE が宣言されているときや実行形式ファイルがダイナミックリンク ライブラリのときには、この警告は出ません。

L4037 *group* : both USE16 and USE32 segments in group; assuming USE32

指定されたグループは、16 ビット セグメントと 32 ビット セグメントの両方から割り当てられています。

L4038 program has no starting address

セグメント化された実行可能プログラムにスタート アドレスがありません。スタート アドレスがないと、通常、プログラムは正しく動作しません。

高水準言語では自動的にスタート アドレスを設定します。C 言語プログラムでは、main 関数にスタート アドレスが置かれます。

アセンブリ言語プログラムでは、END 文でスタート アドレスを指定してください。

MS-DOS プログラムとダイナミックリンク ライブラリ (DLL) では、スタート アドレスがなくてもこのメッセージは表示されません。

L4040 stack size ignored for /TINY

/TINY が指定されていて、かつ、コード セグメントの前にスタック セグメントが定義されていると、LINK はスタック サイズを無視します。

L4042 cannot open old version

モジュール定義 (.DEF) ファイル 中の OLD 命令文に指定されたファイルがオープンできませんでした。

L4043 old version not segmented executable format

モジュール定義 (.DEF) ファイル 中の OLD 命令文に指定されたファイルは、有効なセグメント化された実行可能プログラムではありません。

L4045 name of output file is *filename*

LINK が出力ファイル名を、このファイル名に変更しなければなりませんでした。

ファイル名に拡張子がついていないとき、LINK はデフォルト拡張子 .EXE をとります。Quick ライブラリ、.DLL ファイル、.COM ファイルなどを作ると、LINK は .EXE 以外の拡張子を使います。

/TINY オプション	.COM
/Q オプション	.QLB
.DEF ファイルの LIBRARY 文	.DLL

この警告は、モジュール定義 (.DEF) ファイル の LIBRARY ステートメントで指定した名前が、*exefile* フィールドで指定した名前と異なる場合に起こります。

L4050 file not suitable for /EXEPACK; relink without

パック後のロード イメージとパックのためのオーバーヘッドを合わせたサイズが、パックしていないロード イメージのサイズよりも大きくなります。ファイルをパックする利点がありません。

このメッセージがでたときは、/EXEPACK オプションなしで再度リンクしなおしてください。PWB 内では、[Link Option] の [Additional Debug/Release Option] ダイアログ ボックスにある、[Pack EXE File] チェック ボックスをオフにします。

L4051 *filename* : cannot find library

LINK が指定のライブラリ ファイルを見つけられません。

次のような原因が考えられます。

- 指定のファイルがありません。ライブラリ ファイルの名前か完全なパス指定を入力してください。
- 環境変数 LIB が正しくセットされていません。タイプミス、ディレクトリ リストの間違い、スペース、セミコロン、行末の隠れた文字などをチェックしてください。
- 実行している LINK は古いバージョンです。PATH 環境変数をチェックし、古いリンカを削除するか名前を変えてください。

L4053 VM.TMP : illegal filename

VM.TMP がオブジェクトファイル名として指定されました。

ファイルの名前を変えてからリンカを実行し直してください。

L4054 *filename* : cannot find file

指定したファイルが見つかりません。

新しいファイル名か新しいパス指定のどちらか、または両方を入力してください。

L4055 start address not equal to 0x100 for /TINY

.COM ファイルのスタート アドレスは、16 進数の 100 です。

アセンブリ言語のソース コードのコード セグメントに次のような行を追加してください。

```
ORG 100h
```

L4056 /EXEPACK valid only for OS/2 and real-mode DOS; ignored

/EXEPACK オプションは、Windows プログラムでは指定できません。

L4057 stack specified for DLL; ignored

スタックがダイナミックリンク ライブラリ (DLL) に対して指定されました。コマンド ラインにおける /STACK オプション指定か、モジュール定義 (.DEF) ファイル における STACKSIZE ステートメントが指定されています。LINK は、指定を無視し、スタックを作成しません。

DLL は、スタックを必要としません。

L4058 ignoring alias for already defined symbol *symbol*

指定されたシンボルは、プログラム中で再定義されていますが、OLDNAMES.LIB 内の新しい名前へのエイリアスを持つ C ランタイム ライブラリから参照されています。

LINK は、シンボルに対するエイリアスを無視します。

この警告は、/INFO オプションを指定したときにだけ起こります。

L4067 changing default resolution for weak external *symbol* from *oldresolution* to *newresolution*

矛盾する weak external の外部デフォルトの解決法が見つかりました。最初の解決法を無視し、2 番目のものを使いました。

L4068 ignoring stack size greater than 64K

無効なサイズでスタックが定義されました。LINK は 64K バイトと見なします。

L4069 filename truncated to *filename*

指定するファイル名の長さが、許容範囲を超えました。LINK は、*filename* をファイル名とみなします。

L4070 too many public symbol for sorting

/MAP オプションを指定すると、LINK はスタックと near ヒープ内のすべての利用可能なメモリを使って、パブリック シンボルをソートします。この警告は、パブリック シンボルが、利用可能な領域に入る許容範囲の数を超えている場合に発生します。さらに、シンボルはマップ ファイル内でソートされず、不定の順番でリストされます。

L4076 no segments defined

プログラムにはコードがありません。

この警告は、対象となるファイルがリソースだけで構成されている場合に起こります。

L4077 symbol *function* not defined; ordered allocation ignored

指定された関数は、モジュール定義 (.DEF) ファイル の FUNCTIONS ステートメントで指定されていますが、その関数は定義されていません。

L4079 symbol *function* already defined for ordered allocation; duplicate ignored

指定された関数は、モジュール定義 (.DEF) ファイル の FUNCTIONS ステートメントで 2 回指定されています。

L4080 changing substitute name for alias *symbol*

矛盾するエイリアス名が見つかりました。最初のエイリアスは無視して、2 番目のエイリアスを使いました。

L4081 cannot execute program arguments - message

LINK は、指定されたプログラムを (指定された引数で) 実行できませんでした。

L4082 changing overlay assigment for segment *segment* from *oldnumber* to *newnumber*

指定されたセグメントは、*oldnumber* と *newnumber* で示される 2 つのオーバーレイに割り当てられています。LINK は *newnumber* で示されるオーバーレイに割り当てられているものと見なします。

コマンドラインにおけるカッコ付きオーバーレイの指定と、モジュール定義 (.DEF) ファイル におけるオーバーレイの指定が矛盾している可能性があります。

L4083 changing overlay assigment for symbol *symbol* from *oldnumber* to *newnumber*

指定されたセグメントは、*oldnumber* と *newnumber* で示される 2 つのオーバーレイに割り当てられています。LINK は *newnumber* で示されるオーバーレイに割り当てられているものと見なします。

コマンドラインにおけるカッコ付きオーバーレイの指定と、モジュール定義 (.DEF) ファイル におけるオーバーレイの指定が矛盾している可能性があります。

L4084 *option* : argument missing; option ignored

指定されたオプションには引数が必要ですが、何も指定されていません。

たとえば、次に示すオプションの指定はエラーとなります。

```
/ONERROR
```

L4085 *option* : argument invalid; assuming *argument*

指定されたオプションに与えられた数値引数は、その引数に許された範囲を超えています。LINK は、指定された値を引数だと見なします。

たとえば、次に示す /DYNAMIC:11000 オプション指定は、エラーとなります。

```
/DYNAMIC:11000 : argument invalid; assuming 10922
```

L4086 /r not first on command line; ignored

/r オプションが指定されましたが、LINK コマンドラインの最初の引数ではありませんでした。

/r オプションを指定すると、 LINK が MS-DOS でエクステンデッド メモリを使わずに、 強制的にコンベンショナル メモリだけを使うようにします。

このオプションは、 コマンドラインの最初に指定されなければなりません。応答ファイルや LINK 環境変数でこのオプションを指定することはできません。

Microsoft® Visual Basic™
Programming System for MS-DOS®
リファレンス
1993年2月1日　初版第1刷発行

©1993 Microsoft Corp.

発行　マイクロソフト株式会社

Printed in Japan
落丁・乱丁はお取り替えいたします。

Microsoft®

046-100-903